
TUGAS AKHIR – IF184802

# IMPLEMENTASI ADAPTIF ZONASI PADA *ZONE ROUTING PROTOCOL* BERDASARKAN TINGKAT KEPADATAN *NODE* TETANGGA

HANIA MAGHFIRA
NRP 05111540000042

Dosen Pembimbing I
Ir. F.X. Arunanto, M.Sc.

Dosen Pembimbing II
Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2019


**TUGAS AKHIR – IF184802**

# IMPLEMENTASI ADAPTIF ZONASI PADA *ZONE ROUTING PROTOCOL* BERDASARKAN TINGKAT KEPADATAN *NODE* TETANGGA

**HANIA MAGHFIRA**
**NRP 05111540000042**

**Dosen Pembimbing I**
**Ir. F.X. Arunanto, M.Sc.**

**Dosen Pembimbing II**
**Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.**

**DEPARTEMEN INFORMATIKA**
**Fakultas Teknologi Informasi dan Komunikasi**
**Institut Teknologi Sepuluh Nopember**
**Surabaya 2019**

*(Halaman ini sengaja dikosongkan)*


UNDERGRADUATE THESIS – IF184802

# ADAPTIVE ZONING IMPLEMENTATION IN ZONE ROUTING PROTOCOL BASED ON NEIGHBOR NODE DENSITY LEVEL

**HANIA MAGHFIRA**
**NRP 05111540000042**

**First Advisor**
**Ir. F.X. Arunanto, M.Sc.**

**Second Advisor**
**Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.**

**INFORMATICS DEPARTMENT**
**Faculty of Information Communication and Technology**
**Institut Teknologi Sepuluh Nopember**
**Surabaya 2019**

*(Halaman ini sengaja dikosongkan)*

# LEMBAR PENGESAHAN

## IMPLEMENTASI ADAPTIF ZONASI PADA *ZONE ROUTING PROTOCOL* BERDASARKAN TINGKAT KEPADATAN *NODE* TETANGGA

**TUGAS AKHIR**

Diajukan untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Arsitektur Jaringan Komputer
Program Studi S-1 Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh:
**HANIA MAGHFIRA**
**NRP: 05111540000042**

Disetujui oleh Pembimbing tugas akhir:

1. Ir. F.X. Arunanto, M.Sc. .....
   (NIP. 195701011983031004) (Pembimbing 1)

2. Dr.Eng. Radityo Anggoro, S.Kom., M.Sc. .....
   (NIP. 198410162008121002) (Pembimbing 2)

**SURABAYA**
**JANUARI, 2019**

*(Halaman ini sengaja dikosongkan)*

**IMPLEMENTASI ADAPTIF ZONASI PADA *ZONE ROUTING PROTOCOL* BERDASARKAN TINGKAT KEPADATAN *NODE* TETANGGA**


| | | |
|---|---|---|
| **Nama Mahasiswa** | **:** | **Hania Maghfira** |
| **NRP** | **:** | **05111540000042** |
| **Departemen** | **:** | **Informatika FTIK ITS** |
| **Dosen Pembimbing 1** | **:** | **Ir. F.X. Arunanto, M.Sc.** |
| **Dosen Pembimbing 2** | **:** | **Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.** |


## Abstrak


*Mobile Ad Hoc Network* (MANET) adalah jaringan nirkabel ad hoc yang melakukan konfigurasi mandiri secara kontinyu dan tidak memiliki infrastruktur yang tetap. Salah satu algoritma *routing* pada MANET adalah *Zone Routing Protocol* (ZRP). ZRP merupakan algoritma *routing* hibrid yang menggabungkan kelebihan dari *routing protocol* proaktif dan reaktif, yaitu mengurangi *control overhead* dari *routing protocol* proaktif dan mengurangi *latency* dari *routing protocol* reaktif. Algoritma ZRP membagi *node* ke dalam zona-zona menggunakan radius yang ditentukan secara manual. Penentuan radius secara manual terbilang kurang fleksibel dan optimal mengingat MANET merupakan jaringan ad hoc yang topologinya berubah-ubah dalam periode tertentu.

Modifikasi akan dilakukan pada konsep penentuan zona, yang semula bersifat statis diubah menjadi adaptif berdasarkan tingkat kepadatan *node* tetangga pada zona di dalam radius. Hal ini dilakukan dengan cara menghitung jumlah kepadatan *node* tetangga dari setiap *node*. Jika kepadatan *node* tetangga kurang dari *threshold* yang ditentukan, maka radius akan bertambah satu. Sebaliknya, jika kepadatan *node* tetangga melebihi *threshold* yang ditentukan, maka radius akan berkurang satu.

Hasil dari implementasi menggunakan skenario simulasi NS-2 berupa peningkatan rata-rata *Packet Delivery Ratio* sebesar

2,21 %, peningkatan rata-rata *Routing Overhead* sebesar 18,66 %, dan peningkatan rata-rata *Delivery Delay* sebesar 323 %.

Dari hasil di atas, dapat diketahui bahwa modifikasi zonasi adaptif pada ZRP dapat diimplementasikan dan memengaruhi kinerja algoritma *routing* ZRP pada lingkungan MANET.

**Kata kunci : Kepadatan *Node* Tetangga, MANET, NS-2, ZRP.**

**ADAPTIVE ZONING IMPLEMENTATION IN ZONE ROUTING PROTOCOL BASED ON NEIGHBOR NODE DENSITY LEVEL**

**Student's Name : Hania Maghfira**
**Student's ID : 05111540000042**
**Department : Informatika FTIK-ITS**
**First Advisor : Ir. F.X. Arunanto, M.Sc.**
**Second Advisor : Dr.Eng. Radityo Anggoro, S.Kom, M.Sc.**

## Abstract

*Mobile Ad Hoc Network (MANET) is an ad hoc wireless network that performs continuous self-configuration and does not have a fixed infrastructure. One of the routing algorithms in MANET is Zone Routing Protocol (ZRP). ZRP is a hybrid routing algorithm that combines the advantages of proactive and reactive routing protocols, which reduces the control overhead of proactive routing protocols and reduces the latency of reactive routing protocols [4]. The ZRP algorithm divides nodes into zones using a manually determined radius. Manually determining radius is less flexible and optimal considering MANET is an Ad Hoc network whose topology changes over a period of time.*

*Modifications will be made to the concept of zone determination, which was originally statically converted to adaptive based on the density of neighboring nodes in the zone within the radius. This is done by calculating the density of neighboring nodes from each node. If the density of neighboring nodes is less than the specified threshold, then the radius will increase by one. Conversely, if the density of neighboring nodes exceeds the specified threshold, then the radius will decrease by one.*

*The results of the implementation using the NS-2 simulation scenario in the form of an increase in the average Packet Delivery Ratio of 2,21%, an increase in the Routing Overhead average of 18,66%, and an increase in the average Delivery Delay of 323%.*

*From the results above, it can be seen that modification of adaptive zoning in ZRP can be implemented and affect the performance of the ZRP routing algorithm in the MANET environment.*

***Keyword: Density of Neighbor Node, MANET, NS-2, ZRP.***

# KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Puji syukur kepada Allah Yang Maha Esa atas segala karunia dan rahmat-Nya sehingga penulis dapat menyelesaikan tugas akhir yang berjudul

**"IMPLEMENTASI ADAPTIF ZONASI PADA *ZONE ROUTING PROTOCOL* BERDASARKAN TINGKAT KEPADATAN *NODE* TETANGGA".**

Harapan dari penulis, semoga apa yang tertulis di dalam buku tugas akhir ini dapat bermanfaat bagi pengembangan ilmu pengetahuan saat ini dan ke depannya, serta dapat memberikan kontribusi yang nyata.

Dalam pelaksanaan dan pembuatan tugas akhir ini tentunya sangat banyak bantuan yang penulis terima dari berbagai pihak, tanpa mengurangi rasa hormat penulis ingin mengucapkan terima kasih sebesar-besarnya kepada:

1. Allah SWT. dan Nabi Muhammad SAW. yang telah membimbing penulis selama hidup.
2. Keluarga penulis (Ayah, Ibu, Mbak Fina, Ifa, Gifar, Yusuf, dan keluarga penulis yang lain) yang selalu memberikan dukungan baik berupa doa, moral, dan material yang tak terhingga kepada penulis, sehingga penulis dapat menyelesaikan tugas akhir ini.
3. Bapak Ir. F.X. Arunanto, M.Sc. dan Bapak Dr.Eng. Radityo Anggoro, S.Kom., M.Sc. selaku Dosen Pembimbing penulis yang telah membimbing, memberikan nasihat, dan memotivasi penulis sehingga penulis dapat menyelesaikan tugas akhir ini.
4. Bapak Dr.Eng. Darlis Herumurti, S.Kom., M.Kom. selaku kepala Departemen Informatika ITS.
5. Bapak dan Ibu Dosen yang telah memberikan ilmunya selama penulis berkuliah di Informatika ITS.
6. Teman-teman penulis DaPur Emak Pede (Nahda Fauziyah Zahrah, Purina Qurota Ayunin, dan Anisah Putri Diana) yang selalu memberikan semangat secara tidak langsung kepada

penulis, selalu memberikan hiburan, selalu menemani hari-hari penulis saat senang maupun susah, dan juga menjadi keluarga baru penulis saat berkuliah di Departemen Informatika ITS.

7. Teman-teman KP PLN Hero, Purina, Huda, dan Eritha yang telah menemani kerja praktik penulis dan mau mem-*back up* penulis dikala sakit.
8. Teman-teman penulis Ronald, Rezky, Ufa dan lain-lain yang telah mewarnai kehidupan penulis.
9. Teman-teman dari keluarga besar Laboratorium NCC (Hero, Sisil, Yoga, Dely, Zulfa, Ubut, Zayn, Ical, Azki, Nuza, Akmal, Siraj, Adin, Wasil) yang telah menemani, memberi semangat, doa, serta hiburan dikala penulis sedang jenuh saat pengerjaan tugas akhir ini.
10. Teman-teman dari Kabinet Semangat Berpadu BEM FTIK periode 2017/2018 yang telah mewarnai hari-hari penulis dan mengajarkan makna ikhlas dan sabar, yang sudah berjuang bersama penulis selama kurang lebih satu setengah tahun, yang terkadang membuat penulis kesal namun tidak apa-apa.
11. Teman-teman EA Bahagia BEM FTIf Presisi Bermanfaat yang telah menjadi teman organisasi penulis, menambah rumah baru bagi penulis.
12. Teman-teman Hublu Inspirasi HMTC Inspirasi yang telah menjadi teman berhimpun penulis.
13. Teman-teman Pemandu Phoenix yang telah menjadi rekan mandu penulis selama berkiprah di kepemanduan fakultas.
14. Teman-teman Saman TC yang telah memberikan hiburan, mengajarkan penulis menari saman, dan menjadikan penulis lebih bijak dalam menghadapi adik tingkat.
15. Teman-teman SSL yang telah menjadikan Minggu pagi penulis lebih bermanfaat.
16. Teman-teman pejuang SW 119 yang selalu memberikan informasi penting dan semangat kepada penulis untuk menyelesaikan tugas akhir.

17. Teman-teman angkatan 2015 (Masamalas) yang sudah menjadi saksi hidup perjalanan karir penulis selama berkuliah di Informatika ITS.
18. Untuk orang-orang yang tidak dapat disebutkan satu persatu oleh penulis dan pembaca buku tugas akhir ini.

Penulis telah berusaha sebaik-baiknya dalam menyusun tugas akhir ini. Namun, penulis memohon maaf apabila terdapat kekurangan, kesalahan maupun kelalaian yang telah penulis lakukan. Kritik dan saran yang membangun dapat disampaikan sebagai bahan perbaikan selanjutnya. Tetap semangat dalam menjalani kehidupan, jangan menyerah, karena Allah masih ingin melihat kita berjuang. Semoga kita semua selalu diberi kebahagiaan lahir dan batin dan kesuksesan dunia akhirat. Aamiin.

Surabaya, 14 Januari 2019

Hania Maghfira

*(Halaman ini sengaja dikosongkan)*

# DAFTAR ISI

*(Halaman ini sengaja dikosongkan)*

# DAFTAR GAMBAR

*(Halaman ini sengaja dikosongkan)*

# DAFTAR TABEL

*(Halaman ini sengaja dikosongkan)*

# KODE SUMBER

*(Halaman ini sengaja dikosongkan)*

# BAB I
# PENDAHULUAN

## 1.1 Latar Belakang

Informasi merupakan aset berharga bagi kehidupan manusia. Dengan adanya informasi, manusia dapat mengetahui apa yang terjadi di belahan bumi lain. Informasi juga memegang peranan penting dalam proses komunikasi. Terlebih lagi dalam keadaan genting yang membutuhkan pengiriman dan penerimaan informasi secara cepat. Meningkatnya kebutuhan manusia akan informasi harus diiringi dengan teknologi yang memadai. Salah satu teknologi yang dapat mendukung proses pengiriman informasi dengan cepat adalah *Mobile Ad Hoc Network* (MANET).

MANET merupakan jaringan nirkabel ad hoc yang terdapat pada perangkat bergerak atau *mobile*. Jaringan ad hoc memungkinkan dua perangkat atau lebih untuk berhubungan secara langsung tanpa melalui *router* maupun *access point*. MANET memiliki karakteristik yang dinamis pada *node* dan topologinya. Hal inilah yang akan memengaruhi proses *routing* pada MANET. Algoritma *routing* pada MANET diklasifikasikan menjadi tiga kategori, yaitu proaktif, reaktif, dan hibrid. *Routing* proaktif bersifat *table-driven*, maknanya tiap *node* akan terus memperbarui rute dan daftar tujuan sehingga tabel *routing* akan diperbarui secara periodik. *Routing* reaktif bersifat *on-demand*, maknanya proses pencarian rute hanya akan dilakukan apabila ada permintaan pengiriman data. Sedangkan *routing* hibrid dikembangkan dengan tujuan untuk menggabungkan karakteristik dari algoritma *routing* proaktif dan reaktif untuk mencari rute terbaik sesuai dengan situasi dan kondisi.

Contoh *routing protocol* hibrid adalah *Zone Routing Protocol* (ZRP). Algoritma ZRP ini membagi *node* ke dalam zona-zona menggunakan radius yang ditentukan secara manual. Penentuan radius secara manual terbilang kurang fleksibel dan optimal mengingat MANET merupakan jaringan ad hoc yang topologinya berubah-ubah dalam periode tertentu.

Pada tugas akhir ini akan menggunakan *routing protocol* hibrid ZRP yang akan dimodifikasi pada konsep zonasinya, yang semula bersifat statis diubah menjadi adaptif berdasarkan tingkat kepadatan *node* tetangga.

## 1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini dapat dipaparkan sebagai berikut:

1. Bagaimana cara mengimplementasikan zonasi yang adaptif terhadap tingkat kepadatan *node* tetangga pada algoritma *routing* ZRP?
2. Bagaimana pengaruh zonasi adaptif terhadap kinerja algoritma *routing* ZRP pada lingkungan MANET?

## 1.3 Batasan Permasalahan

Berdasarkan masalah yang diuraikan oleh penulis, maka batasan masalah pada tugas akhir ini adalah:

1. Jaringan yang digunakan adalah MANET.
2. *Routing protocol* yang digunakan adalah *Zone Routing Protocol* (ZRP).
3. Uji coba menggunakan *Network Simulator* 2 (NS-2).

## 1.4 Tujuan

Tujuan dari pembuatan tugas akhir ini adalah untuk mengubah konsep zonasi pada algoritma *routing* ZRP menjadi adaptif terhadap tingkat kepadatan *node* tetangga.

## 1.5 Manfaat

Manfaat yang diperoleh dari pengerjaan tugas akhir ini adalah dapat memberikan informasi tentang dampak dari implementasi zonasi yang adaptif berdasarkan tingkat kepadatan *node* tetangga terhadap kinerja algoritma *routing* ZRP dalam lingkungan MANET.

## 1.6 Metodologi

Pembuatan tugas akhir ini dilakukan dengan menggunakan metodologi sebagai berikut:

### 1.6.1 Penyusunan Proposal Tugas Akhir

Proposal tugas akhir ini berisi gambaran tentang tugas akhir yang akan dibuat. Pendahuluan proposal tugas akhir meliputi hal yang menjadi latar belakang diajukannya usulan tugas akhir, rumusan masalah yang diangkat, batasan masalah yang menjadi konstrain dari tugas akhir, tujuan pembuatan tugas akhir, dan manfaat dari hasil tugas akhir. Di dalam proposal tugas akhir juga dijabarkan mengenai tinjauan pustaka yang menjadi referensi pendukung dalam pembuatan tugas akhir ini. Terdapat pula sub bab jadwal kegiatan yang menjelaskan jadwal pengerjaan tugas akhir.

### 1.6.2 Studi Literatur

Pada studi literatur, akan dilakukan pengumpulan informasi dan referensi yang digunakan dalam pengerjaan tugas akhir yaitu mengenai *Mobile Ad Hoc Network* (MANET), *Zone Routing Protocol* (ZRP), *Network Simulator* 2 (NS-2), dan AWK.

### 1.6.3 Implementasi Sistem

Implementasi merupakan tahap untuk mengimplementasikan metode-metode yang sudah diajukan pada proposal tugas akhir. Untuk membangun algoritma yang telah dirancang sebelumnya, implementasi dilakukan dengan menggunakan NS-2 sebagai simulator jaringan, bahasa C/C++ sebagai bahasa pemrograman untuk uji coba mengimplementasikan metode yang sudah diajukan.

### 1.6.4 Pengujian dan Evaluasi

Pengujian dilakukan dengan menggunakan MANET simulator untuk membuat simulasi keadaan topologi yang diujikan. Simulator yang digunakan adalah NS-2 dan akan menghasilkan keluaran berupa *trace file*. Dari *trace file* tersebut akan dihitung besar *Packet Delivery Ratio* (PDR), *Routing Overhead* (RO), dan

*Delivery Delay* (DD) untuk mengetahui performa kinerja *routing protocol* yang telah dimodifikasi.

### 1.6.5 Penyusunan Buku

Pada tahap ini dilakukan penyusunan buku sebagai dokumentasi dari pelaksanaan tugas akhir yang mencakup seluruh konsep, teori, implementasi, serta hasil yang telah dikerjakan.

## 1.7 Sistematika Penulisan Laporan

Sistematika penulisan laporan tugas akhir adalah sebagai berikut:

1. Bab I. Pendahuluan
   Bab ini berisi penjelasan mengenai latar belakang, rumusan masalah, batasan permasalahan, tujuan, manfaat, metodologi, dan sistematika penulisan dari pembuatan tugas akhir.
2. Bab II. Tinjauan Pustaka
   Bab ini berisi kajian teori atau penjelasan dari metode, algoritma, *library*, dan *tools* yang digunakan dalam penyusunan tugas akhir ini. Kajian teori yang dimaksud berisi tentang penjelasan singkat mengenai *Mobile Ad Hoc Network* (MANET), *Zone Routing Protocol* (ZRP), *Network Simulator* 2 (NS-2), dan AWK.
3. Bab III. Perancangan
   Bab ini berisi pembahasan mengenai perancangan skenario yang akan diimplementasikan dalam tugas akhir. Perancangan skenario berupa perancangan skenario penentuan *threshold*, perancangan penghitungan kepadatan *node* tetangga dan pemilihan radius menggunakan ZRP yang telah dimodifikasi, perancangan simulasi pada NS-2, dan perancangan metrik analisis yang terdiri dari *Packet Delivery Ratio* (PDR), *Routing Overhead* (RO), dan *Delivery Delay* (DD).
4. Bab IV. Implementasi
   Bab ini menjelaskan implementasi yang berbentuk kode sumber dari proses modifikasi protokol ZRP, melakukan simulasi menggunakan NS-2, dan penghitungan metrik analisis.

5. Bab V. Pengujian dan Evaluasi
   Bab ini berisi hasil uji coba dan evaluasi dari implementasi modifikasi pada *routing protocol* ZRP yang telah dilakukan untuk menyelesaikan masalah yang dibahas pada tugas akhir. Pengujian dilakukan dengan skenario yang telah dirancang dan dijalankan di NS-2, yang selanjutnya akan didapatkan hasil untuk dianalisis menggunakan skrip AWK yang nantinya akan menghasilkan metrik analisis berupa PDR, RO, dan DD.
6. Bab VI. Kesimpulan dan Saran
   Bab ini merupakan bab yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan, masalah-masalah yang dialami pada proses pengerjaan tugas akhir, dan saran untuk pengembangan tugas akhir ke depannya.
7. Daftar Pustaka
   Bab ini berisi daftar pustaka yang dijadikan literatur dalam tugas akhir.
8. Lampiran
   Dalam lampiran terdapat kode sumber program secara keseluruhan.

***(Halaman ini sengaja dikosongkan)***

# BAB II
# TINJAUAN PUSTAKA

Bab ini berisi pembahasan mengenai teori-teori dasar atau penjelasan dari metode dan alat yang digunakan dalam tugas akhir. Penjelasan ini bertujuan untuk memberikan gambaran secara umum terhadap program yang dibuat dan berguna sebagai penunjang dalam pengembangan riset yang berkaitan.

## 2.1 MANET

*Mobile Ad Hoc Network* (MANET) adalah jaringan nirkabel ad hoc yang melakukan konfigurasi mandiri secara kontinyu dan tidak memiliki infrastruktur yang tetap. Setiap komponen pada MANET dapat bergerak secara bebas ke segala arah, hal ini menyebabkan perubahan pada topologi secara acak. Karakteristik MANET yang dinamis membuat MANET banyak dimanfaatkan sebagai alat bantu komunikasi dalam situasi yang tidak memiliki infrastruktur seperti upaya rekonstruksi bencana alam, evakuasi Tim SAR, hingga operasi militer [1].

Tipe jaringan MANET adalah desentralisasi, maknanya setiap *node* memiliki tanggung jawab dalam proses pengiriman dan penerimaan paket data. *Node-node* tersebut akan meneruskan paket data ke *node* lain secara dinamis berdasarkan konektivitas jaringan dan algoritma *routing* yang digunakan. Algoritma *routing* pada MANET sendiri diklasifikasikan ke dalam tiga kategori, yaitu *routing protocol* proaktif, reaktif, dan hibrid. *Node-node* pada *routing protocol* proaktif bekerja secara kontinyu untuk mencari rute ke *node* tujuan sehingga tabel *routing* akan selalu ter-*update*. Algoritma *routing protocol* proaktif antara lain *Destination-Sequenced Distance-Vector Routing* (DSDV), *Optimized Link State Routing* (OLSR), *Fisheye State Routing* (FSR), dan *Fuzzy Sighted Link State* (FSLS). Pada *routing protocol* reaktif, *node* baru akan bekerja ketika ada permintaan pengiriman data. Algoritma yang termasuk ke dalam *routing protocol* reaktif antara lain *Dynamic Source Routing* (DSR) dan *Ad Hoc On Demand Distance*

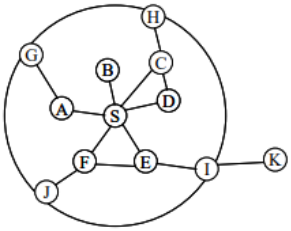
*Vector Routing* (AODV). *Routing protocol* hibrid merupakan kombinasi dari *routing protocol* proaktif dan reaktif. Protokol ini menggunakan klister-klaster untuk menentukan rute dari setiap *node*. *Routing protocol* proaktif digunakan untuk pengiriman paket data antar *node* dalam satu klaster, sedangkan *routing protocol* reaktif digunakan untuk pengiriman paket data antar klaster. Algoritma yang termasuk ke dalam *routing protocol* hibrid salah satunya adalah *Zone Routing Protocol* (ZRP) [2] [3].

## 2.2 *Zone Routing Protocol* (ZRP)

*Zone Routing Protocol* (ZRP) adalah salah satu algoritma *routing* hibrid yang menggunakan parameter berupa *routing* zona. ZRP mulai dikembangkan pada tahun 1997 oleh Haas dan Pearlman. Protokol ini menggabungkan kelebihan dari *routing protocol* proaktif dan reaktif, yaitu mengurangi *control overhead* dari *routing protocol* proaktif dan mengurangi *latency* dari *routing protocol* reaktif.

ZRP membagi jaringannya menjadi dua zona yang berbeda, yaitu zona dalam atau *routing protocol* proaktif lokal yang disebut *IntrA-zone Routing Protocol* (IARP) dan zona luar atau *routing protocol* reaktif global yang disebut *IntEr-zone Routing Protocol* (IERP). Setiap *node* memiliki zona sendiri dan berpotensi berada dalam zona yang tumpang tindih, dan setiap zona berpotensi pula memiliki ukuran yang berbeda-beda. Perbedaan ukuran zona dipengaruhi oleh besarnya radius atau *hop* antar *node*. Penentuan besarnya radius dilakukan secara manual sesuai dengan kebutuhan.

Cara kerja ZRP ditunjukan pada Gambar 2.1, di mana radius yang digunakan adalah dua. *Node* yang termasuk dalam zona *routing node* S adalah semua *node* kecuali *node* K, karena posisi *node* K yang berada diluar zona *routing node* S. Penentuan zona *routing* tidak ditentukan dari jarak fisik, tetapi dari *hop* (lompatan).



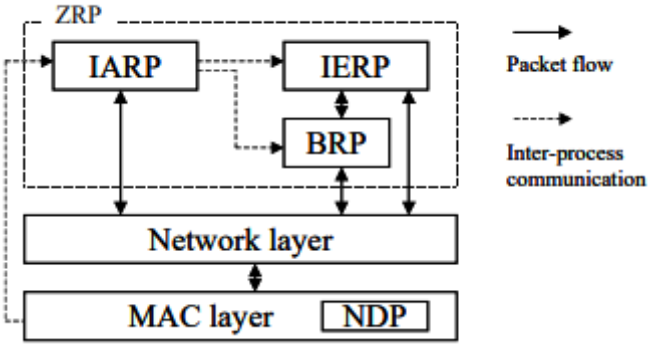

**Gambar 2.1** Contoh Zona *Routing Node* S dengan Radius 2 [4]

Pada zona *routing* terdapat dua jenis *node*, yaitu *peripheral node* dan *interior node*. *Peripheral node* adalah *node* yang memiliki jarak minimum dari *node* sumber sama dengan radius. Sedangkan *interior node* adalah *node* yang memiliki jarak minimum dari *node* sumber kurang dari radius (berada di dalam zona radius). Pada Gambar 2.1, *node* G, H, I, J merupakan *peripheral node*, *node* A, B, C, D, E, F merupakan *interior node*, dan *node* K merupakan *node* di luar zona *routing node* S. Pada Gambar 2.1, terlihat bahwa *node* H dapat dicapai melalui dua jalur, yang pertama sepanjang dua *hop* melalui *node* S-C-H, yang kedua sepanjang tiga *hop* melalui *node* S-D-C-H. Namun, bagaimanapun juga *node* H tetap berada di dalam zona *routing* selama jarak terpendek yang dapat ditempuh adalah kurang dari atau sama dengan zona radius.

Ketika *node* S (*node* sumber) ingin mengirim paket ke *node* tujuan, langkah pertama yang dilakukan adalah mengirimkan permintaan rute ke *peripheral node* miliknya. Permintaan rute berisi alamat sumber, alamat tujuan dan *sequence number*, lalu setiap *peripheral node* akan mengecek zonanya apakah *node* tujuan berada di dalamnya. Jika pada zona *peripheral node* tidak terdapat *node* tujuan, maka *peripheral node* akan menambahkan alamatnya kepada paket *route-request* dan meneruskan paket kepada *peripheral node*. Namun, ketika *node* tujuan berada di dalam zonanya, maka *peripheral node* akan mengirimkan *route-reply* kembali menuju *node* sumber. *Node* sumber menggunakan jalur

yang disimpan dalam paket *route-reply* untuk mengirim paket data ke *node* tujuan.

Arsitektur *routing protocol* ZRP ditunjukkan pada Gambar 2.2. IARP memelihara informasi *routing* dari *node-node* yang berada dalam zona *routing* sebuah *node*. *Route discovery* dan *route maintenance* dilakukan oleh IERP. Bila diperlukan penemuan global, jika topologi zona lokal diketahui, maka hal tersebut bisa digunakan untuk mengurangi lalu lintas. Untuk mengirim *broadcast* paket, ZRP menggunakan konsep *bordercasting*, yang layanannya disediakan oleh *Bordercasting Resolution Protocol* (BRP). BRP menggunakan zona *routing* yang sudah diperluas, yang disediakan oleh IARP lokal untuk membangun *bordercast tree* bersamaan dengan paket permintaan yang diarahkan. BRP menggunakan mekanisme *query control* yang sangat khusus untuk mengarahkan permintaan rute dari area jaringan yang telah dicakup oleh kueri sebelumnya [4].




**Gambar 2.2** Arsitektur ZRP [4]

## 2.3 *Network Simulator 2* (NS-2)

*Network Simulator* (NS) pertama kali dibangun sebagai varian dari REAL *Network Simulator* pada tahun 1989 di UCB (University of California Berkeley). NS sendiri bersifat *open source* di bawah GPL (*Gnu Public License*), sehingga NS dapat

diunduh dan digunakan secara gratis. Sifat *open source* ini juga berdampak pada pengembangan NS menjadi lebih dinamis. Pemodelan media, protokol, komponen jaringan, dan perilaku trafik cukup lengkap bila dibandingkan dengan perangkat lunak lain yang sejenis. Hal ini disebabkan oleh pengembangan NS yang dilakukan oleh banyak periset dunia [5].

*Network simulator* memiliki beberapa versi, salah satunya adalah *Network Simulator* 2 (NS-2). NS-2 merupakan alat bantu simulasi berbasis aktivitas yang banyak digunakan pada penelitian jaringan kabel dan nirkabel. NS-2 menggunakan dua bahasa utama, yaitu C++ dan *Object-oriented Tool Command Language* (OTCL). Bahasa C++ digunakan untuk menggambarkan mekanisme internal (*backend*) pada objek simulasi, sedangkan OTCL digunakan untuk menggambarkan lingkungan simulasi eksternal (*frontend*) pada perakitan dan konfigurasi objek simulasi. Hasil dari simulasi menggunakan NS-2 berupa *trace file* (*.tr) dan *network animator* (*.nam) [6] [7].

Pada tugas akhir ini digunakan NS-2 versi 2.35 untuk melakukan simulasi lingkungan MANET menggunakan *routing protocol* ZRP yang sudah dimodifikasi. *Trace file* yang dihasilkan oleh NS-2 juga digunakan sebagai informasi untuk mengukur performa *routing protocol* ZRP yang sudah dimodifikasi.

### 2.3.1 Instalasi

Sebelum melakukan instalasi NS-2, terlebih dahulu harus memastikan bahwa *compiler* yang terpasang sudah sesuai dengan standar yang digunakan oleh NS-2, yaitu gcc-4.8. Untuk memasang *compiler* tersebut dapat dilakukan dengan perintah sebagai berikut:

```
sudo apt install gcc-4.8 g++-4.8
```

NS-2 membutuhkan beberapa *package* yang harus sudah terpasang pada perangkat komputer sebelum memulai instalasi NS-

2. Untuk memasang *dependency* yang dibutuhkan dapat dilakukan dengan perintah sebagai berikut :

```
sudo apt update

sudo apt-get install build-essential
autoconf automake libxmu-dev
```

Kemudian, unduh NS-2 versi 2.35 yang dapat dilakukan melalui tautan berikut :

```
https://sourceforge.net/projects/nsnam/fi
les/allinone/ns-allinone-2.35/ns-
allinone-2.35.tar.gz/download
```

lalu ekstrak berkas unduhan NS-2 pada direktori yang diinginkan. Selanjutnya, masuk ke dalam direktori ns-allinone-2.35/ns-2.35, lalu ubah baris kode ke-137 pada *file* ls.h di folder linkstate menjadi seperti perintah berikut :

```
gedit Makefile.in
```
ubah baris kode ke-36 dan 37 menjadi:
```
CC    = gcc-4.8
CPP   = g++-4.8

gedit linkstate/ls.h
```
ubah baris kode ke-137 menjadi:
```
void eraseAll(){this->erase
(baseMap::begin(),baseMap::end());}
```

Setelah mengubah baris kode, kembali ke direktori ns-allinone-2.35 dan instal NS-2 dengan menjalankan perintah berikut:

```
./install
```

Kemudian, *environment* sistem pada NS-2 diatur agar NS-2 dapat dijalankan. Pengaturan tersebut dapat dilakukan melalui perintah sebagai berikut:

```
sudo gedit ~/.bashrc

Copy skrip berikut diakhir file:
# LD_LIBRARY_PATH
OTCL_LIB=/home/hania/ns-allinone-
2.35/otcl-1.14
NS2_LIB=/home/hania/ns-allinone-2.35/lib
X11_LIB=/usr/X11R6/lib
USR_LOCAL_LIB=/usr/local/lib
export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$OTCL_LI
B:$NS2_LIB:$X11_LIB:$USR_LOCAL_LIB

# TCL_LIBRARY
TCL_LIB=/home/hania/ns-allinone-
2.35/tcl8.5.10/library
USR_LIB=/usr/lib
export TCL_LIBRARY=$TCL_LIB:$USR_LIB

# PATH
XGRAPH=/home/hania/ns-allinone-
2.35/bin:/home/hania/ns-allinone-
2.35/tcl8.5.10/unix:/home/hania/ns-
allinone-2.35/tk8.5.10/unix
NS=/home/hania/ns-allinone-2.35/ns-2.35/
NAM=/home/hania/ns-allinone-2.35/nam-
1.15/
PATH=$PATH:$XGRAPH:$NS:$NAM
```

Lalu masukkan perintah `source ~/.bashrc` dan `ns` pada terminal.

### 2.3.2 *Trace File*

*Trace file* merupakan dokumen hasil simulasi NS-2 yang berisikan informasi detail mengenai pengiriman paket data. *Trace file* digunakan untuk menganalisis performa *routing protocol* yang disimulasikan. Penjelasan mengenai *trace file* ditunjukkan pada Tabel 2.1.

**Tabel 2.1** Detail Penjelasan *Trace File* ZRP

| **Kolom ke-** | **Penjelasan** | **Isi** |
|---|---|---|
| 1 | *Event* | s: *sent*<br>r: *received*<br>f: *forwarded*<br>D: *dropped* |
| 2 | *Time* | Waktu terjadinya *event* |
| 3 | ID *Node* | _x_: dari 0 hingga banyak *node* pada topologi |
| 4 | *Layer* | AGT: *application*<br>RTR: *routing*<br>LL: *link layer*<br>IFQ: *packet queue*<br>MAC: MAC<br>PHY: *physical* |
| 5 | *Flag* | ---: Tidak ada |
| 6 | *Sequence Number* | Nomor paket |
| 7 | Tipe Paket | ZRP: paket *routing ZRP*<br>Cbr: berkas paket CBR (*Constant Bit Rate*)<br>RTS: *Request To Send* yang dihasilkan MAC 802.11<br>CTS: *Clear To Send* yang dihasilkan MAC 802.11<br>ACK: MAC ACK<br>ARP: Paket *link layer Address Resolution Protocol* |
| 8 | Ukuran | Ukuran paket pada *layer* saat itu |

| Kolom ke- | Penjelasan | Isi |
|---|---|---|
| 9 | Detail MAC | [a b c d]<br>a: perkiraan waktu paket<br>b: alamat penerima<br>c: alamat asal<br>d: IP *header* |
| 10 | *Flag* | ------: Tidak ada |
| 11 | Detail IP *source*, *destination*, dan *nexthop* | [a:b c:d e f]<br>a: IP *source node*<br>b: *port source node*<br>c: IP *destination node* (jika -1 berarti *broadcast*)<br>d: *port destination node*<br>e: IP *header* ttl<br>f: IP *nexthop* (jika 0 berarti *node* 0 atau *broadcast)* |

## 2.4 AWK

AWK merupakan sebuah program filter untuk teks berupa bahasa pemrograman pada *shell* atau C yang memiliki karakteristik sebagai alat untuk melakukan filter atau manipulasi data. AWK bersifat *data-driven* yang berisikan kumpulan perintah yang akan dijalankan pada data tekstural baik secara langsung pada *file* atau digunakan sebagai bagian dari *pipeline*. Secara umum AWK dapat digunakan untuk mengelola *database* sederhana, membuat laporan, memvalidasi data, dan membuat algoritma yang digunakan untuk mengubah bahasa komputer ke bahasa lainnya. Dengan kata lain, AWK menyediakan fasilitas yang dapat memudahkan untuk memecah bagian data untuk proses selanjutnya, mengurutkan data, dan menampilkan komunikasi jaringan yang sederhana. Pada tugas akhir ini, AWK digunakan untuk membuat skrip menghitung metrik analisis berupa *Packet Delivery Ratio* (PDR), *Routing Overhead* (RO), dan *Delivery Delay* (DD) dari hasil simulasi menggunakan NS-2 [8].

***(Halaman ini sengaja dikosongkan)***

# BAB III
# PERANCANGAN

Bab ini membahas mengenai perancangan implementasi sistem yang dibangun pada tugas akhir. Bagian yang akan dijelaskan pada bab ini berawal dari deskripsi umum sistem, perancangan skenario, alur, hingga gambaran implementasi sistem yang diterapkan pada *Network Simulator* 2 (NS-2).

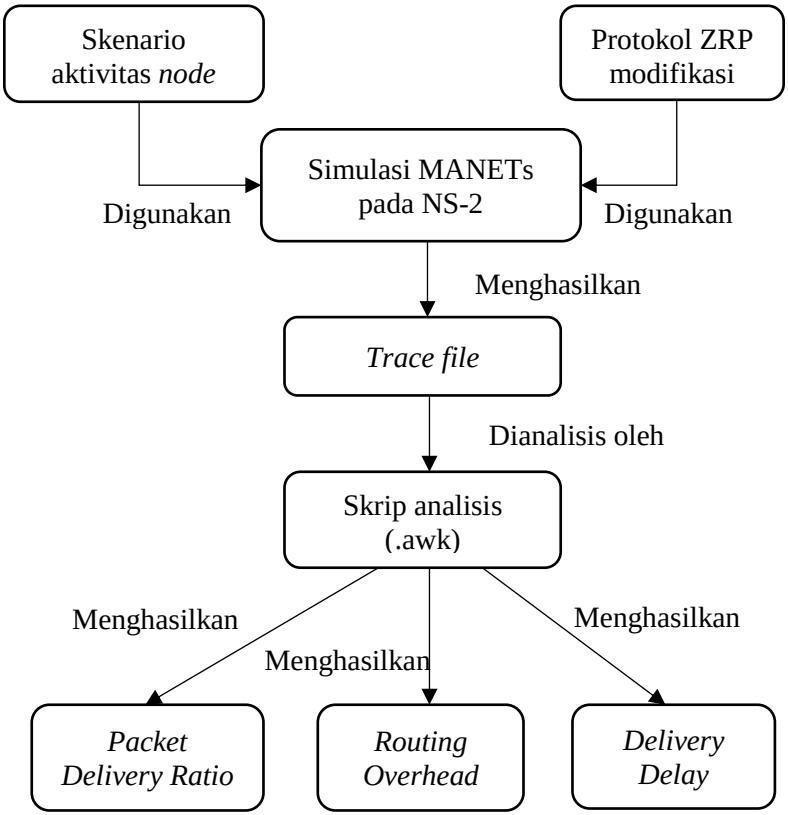
## 3.1 Deskripsi Umum

Pada tugas akhir ini akan dilakukan implementasi *routing protocol* ZRP yang dimodifikasi dengan menerapkan metode zonasi yang adaptif terhadap tingkat kepadatan *node* tetangga. Pembuatan skenario MANET ini menggunakan simulator yaitu *Network Simulator* 2 (NS-2).

Perancangan skenario uji coba diawali dengan menetapkan *threshold* kepadatan *node* tetangga. *Threshold* ini digunakan sebagai batas maksimal jumlah kepadatan *node* tetangga yang boleh dimiliki *node* saat simulasi dijalankan. Selanjutnya, melakukan inisialisasi nilai radius yang akan digunakan sebagai nilai awal saat simulasi dijalankan, dan penentuan nilai radius maksimal yang boleh digunakan dalam simulasi. Nilai awal dan maksimal radius ini bersifat tetap. Kemudian melakukan modifikasi pemilihan radius yang akan digunakan untuk proses simulasi. Jika kepadatan *node* tetangga pada zona di dalam radius (zona IARP) kurang dari *threshold* dan besar radius kurang dari radius maksimal, maka radius akan bertambah satu. Sebaliknya, jika kepadatan *node* tetangga melebihi *threshold* yang ditentukan, maka radius akan berkurang satu.

Skenario pengiriman paket data menggunakan *routing protocol* ZRP akan berjalan setelah besar radius terpilih. Simulasi yang dilakukan akan menghasilkan *trace file*, yang kemudian dianalisis menggunakan skrip AWK untuk mendapatkan *Packet Delivery Ratio* (PDR), *Routing Overhead* (RO), dan *Delivery Delay* (DD). Analisis tersebut dapat mengukur performa *routing*

*protocol* ZRP yang telah dimodifikasi dibandingkan dengan *routing protocol* ZRP asli. Diagram rancangan simulasi dengan *routing protocol* ZRP yang telah dimodifikasi dapat dilihat pada Gambar 3.1.




**Gambar 3.1** Diagram Rancangan Simulasi ZRP Modifikasi

## 3.2 Daftar Istilah

Daftar istilah yang sering digunakan pada buku tugas akhir ini dapat dilihat pada Tabel 3.1.

**Tabel 3.1** Daftar Istilah

| No. | Istilah | Penjelasan |
|---|---|---|
| 1 | *Mobile Adhoc Network* (MANET) | *Mobile Ad Hoc Network* (MANET) merupakan jaringan nirkabel ad hoc yang melakukan konfigurasi mandiri secara kontinyu dan tidak memiliki infrastruktur yang tetap. |
| 2 | *Zone Routing Protocol* (ZRP) | *Zone Routing Protocol* (ZRP) merupakan salah satu algoritma *routing* hibrid yang menggunakan parameter berupa *routing* zona dan menggabungkan kelebihan dari *routing protocol* proaktif dan reaktif. |
| 3 | *Network Simulator* 2 (NS-2) | NS-2 merupakan alat bantu simulasi berbasis aktivitas pada penelitian jaringan kabel maupun nirkabel. |
| 4 | *Packet Delivery Ratio* (PDR) | PDR merupakan teknik penghitungan perbandingan jumlah paket yang diterima oleh *node* tujuan dengan jumlah paket yang dikirim oleh *node* sumber. |
| 5 | *Routing Overhead* (RO) | RO merupakan teknik penghitungan jumlah *routing* paket kontrol yang ditransmisikan ke *node* tujuan selama simulasi terjadi. |
| 6 | *Delivery Delay* (DD) | DD merupakan teknik penghitungan waktu yang diperlukan mulai dari paket dikirimkan oleh *node* sumber sampai paket data tersebut berhasil diterima *node* tujuan. |
| 7 | *Threshold* | Batas nilai yang dijadikan sebagai acuan. |
| 8 | *Node* tetangga | *Node* di sekitar *node* sumber yang berada di dalam zona *IntrA-zone Routing Protocol* (IARP). |
| 9 | Kepadatan *Node* Tetangga | Perbandingan jumlah *node* tetangga dengan jumlah seluruh *node*. |

## 3.3 Perancangan Skenario *Threshold* Kepadatan *Node* Tetangga

Perancangan skenario *threshold* kepadatan *node* tetangga pada tugas akhir ini dilakukan melalui pra-uji coba menghitung nilai metrik analisis untuk setiap besaran kepadatan *node* tetangga. Besar kepadatan *node* tetangga yang digunakan adalah 0 hingga 1. Simulasi yang dilakukan menggunakan ZRP yang telah dimodifikasi. Hasil dari penghitungan metrik analisis nantinya akan dilihat nilai metrik analisis yang tertinggi, dengan nilai PDR yang menjadi parameter utama penentuan *threshold*. Besar kepadatan *node* tetangga yang memiliki nilai PDR paling tinggi akan dijadikan sebagai nilai *threshold* untuk semua skenario simulasi.

## 3.4 Analisis dan Perancangan Modifikasi *Routing Protocol* ZRP

Perancangan modifikasi *routing protocol* pada tugas akhir ini mencakup pada perancangan penghitungan kepadatan *node* tetangga untuk setiap radius dan perancangan pemilihan radius yang adaptif terhadap jumlah kepadatan *node* tetangga.

### 3.4.1 Perancangan Penghitungan Kepadatan *Node* Tetangga untuk Setiap Radius

Ketika nilai *threshold* kepadatan *node* tetangga serta nilai radius awal dan maksimal telah ditentukan, maka ZRP yang telah dimodifikasi akan menghitung kepadatan *node* tetangga. Kepadatan *node* tetangga yang dimaksud adalah kepadatan *node* di sekitar *node* sumber yang berada di dalam zona IARP.

Langkah awal untuk menghitung kepadatan *node* tetangga adalah dengan menghitung terlebih dahulu jumlah *node* di sekitar *node* sumber yang berada di dalam zona IARP. Setelah itu, membagi jumlah *node* tetangga dengan jumlah seluruh *node* yang digunakan pada simulasi. Sehingga, akan didapat besar kepadatan *node* tetangga di dalam zona IARP. *Pseudocode* untuk

penghitungan kepadatan *node* tetangga dapat dilihat pada Gambar 3.2.

```
total_node_IARP = jumlah node tetangga

density =
total_node_IARP/total_node_simulasi
```

**Gambar 3.2** *Pseudocode* Penghitungan Kepadatan *Node* Tetangga

### 3.4.2 Perancangan Pemilihan Radius

Pemilihan radius akan dimulai setelah kepadatan *node* tetangga diketahui. Selanjutnya, jumlah kepadatan *node* tetangga dibandingkan dengan nilai *threshold* yang telah ditentukan. Jika jumlah kepadatan *node* tetangga kurang dari nilai *threshold* dan besar radius kurang dari besar radius maksimal, maka besar radius bertambah satu. Sebaliknya, jika kepadatan *node* tetangga melebihi *threshold* yang ditentukan dan besar radius lebih dari radius minimal, maka radius akan berkurang satu. *Pseudocode* untuk penghitungan radius dapat dilihat pada Gambar 3.3.

```
if (density < threshold && radius <
radius_max)
then
      radius++

if (density > threshold && radius >
radius_min)
then
      radius--
```

**Gambar 3.3** *Pseudocode* Penghitungan Radius

## 3.5 Perancangan Simulasi pada NS-2

Simulasi MANET pada NS-2 dilakukan dengan menggunakan skrip TCL dengan parameter yang berisi konfigurasi untuk membangun lingkungan simulasi. Parameter simulasi perancangan sistem MANET dapat dilihat pada Tabel 3.2 [9].

**Tabel 3.2** Parameter Lingkungan Simulasi

| No. | Parameter | Spesifikasi |
|---|---|---|
| 1. | *Network Simulator* | NS-2 versi 2.35 |
| 2. | *Routing protocol* | ZRP |
| 3. | Radius | radius awal = 1,<br>radius maksimal = 6 |
| 4. | Waktu Simulasi | 200 detik |
| 5. | Waktu Pengiriman Paket Data | 2,5 – 200 detik |
| 5. | Area Simulasi | 1500 m x 1500 m |
| 6. | Banyak *Node* | 25, 50, 100 |
| 7. | Kecepatan Pergerakan *Node* | 5 m/s |
| 8. | Radius Transmisi | 250 m |
| 9. | Tipe Koneksi | UDP |
| 10. | Tipe Data | *Constant Bit Rate* (CBR) |
| 11. | *Source/Destination* | Dinamis |
| 12. | Kecepatan Generasi Paket | 1 paket per detik |
| 13. | Ukuran Paket Data | 512 *bytes* |
| 14. | Protokol MAC | IEEE 802.11 |
| 15. | Tipe Antena | OmniAntenna |
| 16. | Tipe Interface Queue | Droptail/PreQueue |
| 17. | Tipe Kanal | *Wireless Channel* |
| 18. | Tipe Trace | Old Format Trace |

## 3.6 Perancangan Metrik Analisis

Berikut ini merupakan parameter-parameter yang akan dianalisis pada tugas akhir ini untuk mengetahui pengaruh dari implementasi zonasi yang adaptif terhadap tingkat kepadatan *node* tetangga pada *routing protocol* ZRP.

### 3.6.1 *Packet Delivery Ratio* (PDR)

*Packet Delivery Ratio* (PDR) merupakan teknik penghitungan perbandingan antara jumlah paket yang diterima oleh *node* tujuan dengan jumlah paket yang dikirim oleh *node* sumber. PDR dihitung dengan persamaan 3.1.

$$PDR = \frac{packet\ received}{packet\ sent} x\ 100\ \%\ \textbf{(3.1)}$$

Pada persamaan 3.1, didapatkan informasi bahwa :
***Packet Received*** : jumlah total paket yang diterima oleh *node* tujuan.
***Packet Sent*** : jumlah total paket yang dikirim oleh *node* sumber.

### 3.6.2 *Routing Overhead* (RO)

*Routing Overhead* (RO) merupakan teknik penghitungan jumlah *routing* paket kontrol yang ditransmisikan per data paket ke *node* tujuan selama simulasi terjadi. *Routing* paket kontrol yang ditransmisikan berupa paket *route request* (RREQ), *route reply* (RREP), dan *route error* (RERR). Penghitungan RO digunakan untuk melihat banyaknya rute yang dicari dalam jaringan selama simulasi dijalankan. RO dihitung dengan persamaan 3.2.

$$RO = \sum_{m=1}^{sent\ and\ forwarded\ num} packet\ sent \quad \textbf{(3.2)}$$

### 3.6.3 *Delivery Delay* (DD)

*Delivery Delay* (DD) merupakan teknik penghitungan waktu yang diperlukan mulai dari paket dikirimkan oleh *node* sumber sampai paket data tersebut berhasil diterima oleh *node* tujuan. Penghitungan DD digunakan untuk melihat berapa lama waktu yang diperlukan paket untuk sampai ke *node* tujuan. DD dihitung dengan persamaan 3.3.

$$DD = Time\ packet\ received - Time\ packet\ sent \quad \textbf{(3.3)}$$

Pada persamaan 3.3, didapatkan informasi bahwa :
***Time Packet Received*** : waktu ketika paket diterima oleh *node* tujuan.
***Time Packet Sent*** : waktu ketika paket dikirim oleh *node* sumber.

***(Halaman ini sengaja dikosongkan)***

# BAB IV
# IMPLEMENTASI

Bab ini membahas mengenai implementasi sistem yang dilakukan berdasarkan rancangan yang telah dijabarkan pada bab sebelumnya. Implementasi berupa kode sumber untuk membangun program. Implementasi yang dijelaskan meliputi lingkungan pembangunan sistem, implementasi skenario *threshold* kepadatan *node* tetangga, implementasi modifikasi *routing protocol* ZRP, implementasi simulasi pada NS-2, dan implementasi metrik analisis.

## 4.1 Lingkungan Pembangunan Sistem

Lingkungan sistem yang digunakan untuk membangun implementasi skenario dapat dilihat pada Tabel 4.1.

**Tabel 4.1** Tabel Lingkungan Pembangunan Sistem

| Perangkat | Komponen | Spesifikasi |
|---|---|---|
| **Perangkat Keras** | *Processor* | Intel(R) Core(TM) i5-7400 CPU @ 3.00GHz 3.00 GHz |
| | Memori | RAM 8,00 GB |
| | Penyimpanan | 928 GB |
| **Perangkat Lunak** | Sistem Operasi | Ubuntu Bionic 18.04 LTS 64 bit |
| | Simulator | *Network Simulator* 2 versi 2.35 |

## 4.2 Instalasi *Routing Protocol* ZRP pada NS-2

Pada umumnya, *routing protocol* ZRP tidak tersedia pada bawaan instalasi NS-2 versi 2.35 standar. Maka dari itu, perlu dilakukan instalasi *patch routing protocol* ZRP agar *routing protocol* ZRP dapat dijalankan pada NS-2. Sebelum melakukan instalasi, pastikan bahwa NS-2 sudah diinstal pada perangkat.

Langkah-langkah instalasi *routing protocol* ZRP adalah sebagai berikut :

1. Unduh *patch* ZRP pada tautan berikut http://bit.ly/PatchZRP-NS235.
2. Masuk ke dalam direktori *ns-allinone-2.35.*
3. Salin *file patch* yang diunduh tadi (*file* bernama *zrp-ns235.patch*) ke dalam direktori *ns-allinone-2.35.*
4. Setelah disalin, jalankan perintah berikut untuk melakukan *patch routing protocol* ZRP.

```
patch -p0 < zrp-ns235.patch
```

5. Setelah melakukan *patch*, lakukan instalasi pada direktori *ns-allinone-2.35* menggunakan perintah berikut.

```
./install
```

6. Atur *environment variables* pada .bashrc agar NS-2 dapat dijalankan pada semua lingkungan Linux.

## 4.3 Implementasi Modifikasi *Routing Protocol* ZRP

*Routing protocol* ZRP merupakan salah satu *routing protocol* hibrid yang umum digunakan dalam simulasi MANET. *Routing protocol* ZRP ini membagi *node* ke dalam zona-zona menggunakan radius yang ditentukan secara manual. Penentuan radius secara manual terbilang kurang fleksibel mengingat MANET merupakan jaringan ad hoc yang topologinya berubah-ubah dalam periode tertentu.

Pada tugas akhir ini dilakukan modifikasi pada *routing protocol* ZRP untuk konsep zonasinya, di mana nilai radius akan berubah dinamis berdasarkan tingkat kepadatan *node* tetangga sesuai dengan *threshold* yang telah ditentukan. Sehingga, pada tugas akhir ini dapat dilihat performa pada *routing protocol* ZRP yang telah dimodifikasi.

Implementasi modifikasi *routing protocol* ZRP ini dibagi menjadi dua bagian, yaitu

1. Implementasi penghitungan kepadatan *node* tetangga untuk setiap radius, dan
2. Implementasi pemilihan radius.

Kode program implementasi *routing protocol* ZRP pada NS-2 versi 2.35 terletak di dalam direktori *ns-2.35/zrp*. Pada direktori tersebut terdapat beberapa dokumen seperti *constants.h*, *zrp.cc*, *zrp.h*, dan *zrp.o*. Pada tugas akhir ini, penulis hanya melakukan modifikasi pada dokumen *zrp.cc*, *zrp.h*, dan *constants.h* untuk menghitung jumlah kepadatan *node* tetangga dan memilih radius sesuai dengan tingkat kepadatan *node* tetangganya. Pada bagian ini, penulis akan menjelaskan langkah-langkah dalam mengimplementasikan modifikasi *routing protocol* ZRP agar radius menjadi adaptif terhadap tingkat kepadatan *node* tetangga.

### 4.3.1 Implementasi Penghitungan Kepadatan *Node* Tetangga untuk Setiap Radius

Langkah awal yang dilakukan untuk menghitung jumlah kepadatan *node* tetangga seperti yang telah dirancang pada subbab 3.4.1 adalah dengan menghitung jumlah *node* tetangga yang berada di dalam zona IARP.

Pada *routing protocol* ZRP, terdapat fungsi *print_tables* dalam kelas *NDPAgent* yang berfungsi untuk mencatat *node* tetangga yang berjarak satu *hop* dari *node* sumber. Pada tugas akhir ini, penulis memodifikasi fungsi tersebut agar bisa mengeluarkan informasi berisi *node* tetangga yang berjarak satu *hop* dari *node* sumber. Potongan kode modifikasi dapat dilihat pada Kode Sumber 4.1.

```
int NDPAgent::print_tables() {
   ...
   return neighborLst_.numNeighbors_;
}
```

**Kode Sumber 4.1** Modifikasi Fungsi *print_tables* dalam Kelas *NDPAgent*

Dalam *routing protocol* ZRP, terdapat juga kelas *IARPAgent* yang berisi fungsi-fungsi untuk mengatur proses pada IARP ketika simulasi dijalankan. Pada tugas akhir ini, penulis memanfaatkan kelas tersebut untuk membuat fungsi baru guna menghitung jumlah *node* yang berada di dalam zona IARP, yang jaraknya lebih dari satu *hop* dari *node* sumber. Fungsi tersebut penulis beri nama *count_node_in_rad* dan diletakkan di dalam dokumen *zrp.cc*. Potongan kode modifikasi dapat dilihat pada Kode Sumber 4.2.

```
int IARPAgent::count_node_in_rad(){
   InnerRoute *cur = irLst_.head_;
   int result = 0;
   printf("\nIARP Routes: %d, [ ", agent_-
   >myaddr_);
   if(irLst_.numRoutes_ == 0){
       printf("EMPTY");
   }
   for(int i=0; i<irLst_.numRoutes_; i++){
       if(cur->numHops_<= agent_->radius_){
           printf("(%d:%d), ", cur->addr_, cur-
           >nextHop_);
           result++;
       }
       cur = cur->next_;
   }
   printf("]");
   return result;
}
```

**Kode Sumber 4.2** Menghitung *Node* di dalam IARP

Setelah melakukan modifikasi pada fungsi *print_tables* dan membuat fungsi baru, selanjutnya penulis melakukan modifikasi pada kelas *NDPAgent* dan *IARPAgent* yang terletak pada dokumen *zrp.h*. Pada kelas *NDPAgent*, modifikasi yang dilakukan adalah mengganti tipe data fungsi *print_tables* menjadi integer. Potongan

kode modifikasi dapat dilihat pada Kode Sumber 4.3. Kode selengkapnya dapat dilihat pada lampiran 1.

```
class NDPAgent{
  public:
       ...
       int print_tables();
}
```

**Kode Sumber 4.3** Inisialisasi Fungsi *print_tables* dalam Kelas *NDPAgent*

Sedangkan pada kelas *IARPAgent*, modifikasi yang dilakukan adalah menambah fungsi yang baru dibuat agar fungsi dapat dikenali. Potongan kode modifikasi dapat dilihat pada Kode Sumber 4.4. Kode selengkapnya dapat dilihat pada lampiran 2.

```
class IARPAgent{
  public:
       ...
       int count_node_in_rad();
}
```

**Kode Sumber 4.4** Inisialisasi Fungsi *count_node_in_rad*

Penulis juga melakukan modifikasi pada dokumen *constant.h* dengan mengubah nilai DEBUG menjadi 1 agar jumlah *node* tetangga dapat dikeluarkan pada dokumen lain. Potongan kode modifikasi dapat dilihat pada Kode Sumber 4.5. Kode selengkapnya dapat dilihat pada lampiran 3.

```
...
#define DEBUG 1
...
```

**Kode Sumber 4.5** Modifikasi Dokumen *constant.h*

Pada *routing protocol* ZRP, terdapat kelas *NDPAckTimer* yang terletak pada dokumen *zrp.cc*. Fungsi tersebut digunakan untuk melakukan *update node* tetangga yang berjarak satu *hop* dari *node* sumber secara periodik. Apabila sebuah *node* tidak menerima ACK dari *node* tetangga yang berjarak satu *hop* darinya, maka *node* tersebut dihapus dari tabel *node* tetangga dan kelas *NDPAckTimer* akan memberitahu kelas *IARP* untuk melakukan *update* rute. Di dalam fungsi *NDPAckTimer*, penulis memanggil fungsi *print_tables* yang telah dimodifikasi dan menyimpannya ke dalam variabel jumlah *node* tetangga yang berjarak satu *hop*. Selanjutnya, penulis menambah kode untuk menghitung kepadatan *node* tetangga, yaitu dengan membagi jumlah *node* tetangga dengan jumlah seluruh *node* yang digunakan dalam simulasi. Potongan kode modifikasi untuk menghitung kepadatan *node* tetangga dapat dilihat pada Kode Sumber 4.6. Kode selengkapnya dapat dilihat pada lampiran 4.

```
void NDPAckTimer::handle(Event* e){
   ...

   if (agent_->radius_==1) {
      int num_neighbor = (agent_-> ndpAgt_)
                          .print_tables();

      float density = (float)num_neighbor /
                       jumlah_node_simulasi;

   }
}
```

**Kode Sumber 4.6** Memanggil Fungsi *print_tables* dalam Kelas *NDPAckTimer*

Pada *routing protocol* ZRP, terdapat juga kelas *IARPPeriodicUpdateTimer* yang terletak pada dokumen *zrp.cc*. Fungsi tersebut digunakan untuk melakukan *update* IARP secara

periodik. Di dalam fungsi ini, penulis memanggil fungsi yang telah dibuat dan menyimpannya ke dalam variabel jumlah *node* tetangga. Selanjutnya, penulis juga menambah kode untuk menghitung kepadatan *node* tetangga, yaitu dengan membagi jumlah *node* tetangga dengan jumlah seluruh *node* yang digunakan dalam simulasi. Potongan kode modifikasi untuk menghitung kepadatan *node* tetangga dapat dilihat pada Kode Sumber 4.7. Kode selengkapnya dapat dilihat pada lampiran 5.

```
void IARPPeriodicUpdateTimer::handle(Event*
e){
   ...

   int node_in_rad = (agent_-> iarpAgt_)
                      .count_node_in_rad()-1;

   float kepadatan = (float)node_in_rad /
                      jumlah_node_simulasi;
   ...
}
```

**Kode Sumber 4.7** Memanggil Fungsi *count_node_in_rad* dalam Kelas *IARPPeriodicUpdateTimer*

### 4.3.2 Implementasi Pemilihan Radius

Langkah awal yang dilakukan untuk memilih radius secara adaptif terhadap kepadatan *node* tetangga seperti yang telah dirancang pada subbab 3.4.2 adalah membandingkan kepadatan *node* tetangga dengan nilai *threshold*. Proses membandingkan kepadatan *node* tetangga dilakukan pada kelas *NDPAckTimer* dan *IARPPeriodicUpdateTimer* dalam dokumen *zrp.cc*.

Pada kelas *NDPAckTimer*, apabila kepadatan *node* tetangga kurang dari *threshold*, maka radius bertambah satu. Penambahan baris kode diletakkan di dalam baris kode untuk menghitung kepadatan *node* tetangga. Potongan kode untuk proses pemilihan radius dapat dilihat pada Kode Sumber 4.8. Kode selengkapnya dapat dilihat pada lampiran 4.

```
void NDPAckTimer::handle(Event* e){
   ...

   if (agent_->radius_==1) {
      ...

      float th = nilai_threshold;

      printf("\nNDP UPDATED FOR NODE NUMBER :
             %d, radius : %d, kepadatan : %f,
             num_neighbor : %d\n\n", agent_-
             >myaddr_, agent_->radius_,
             density, num_neighbor);

      if (density < th && num_neighbor > 0) {
          agent_->radius_ +=1;
      }
   }
}
```

**Kode Sumber 4.8** Pemilihan Radius dalam Kelas *NDPAckTimer*

Selanjutnya pada kelas *IARPPeriodicTimer*, terdapat dua kondisi dalam implementasi radius yang adaptif. Kondisi pertama, yaitu apabila kepadatan *node* tetangga kurang dari *threshold* serta besar radius lebih dari satu dan kurang dari radius maksimal, maka radius bertambah satu. Kondisi kedua, yaitu apabila kepadatan *node* tetangga lebih dari *threshold* dan besar radius lebih dari satu, maka radius akan bekurang satu. Penambahan baris kode diletakkan tepat setelah penghitungan kepadatan *node* tetangga. Potongan kode untuk proses pemilihan radius dapat dilihat pada Kode Sumber 4.9. Kode selengkapnya dapat dilihat pada lampiran 5.

```
void IARPPeriodicUpdateTimer::handle(Event*
e){
   ...

 int node_in_rad = (agent_-> iarpAgt_)
                     .count_node_in_rad()-1;

 float kepadatan = (float)node_in_rad /
                     jumlah_node_simulasi;

 float treshold = nilai_threshold;

 printf("\nIARP UPDATED FOR NODE NUMBER : %d,
         radius : %d, kepadatan : %f,
         node_in_rad : %d\n\n", agent_-
         >myaddr_, agent_->radius_,
         kepadatan, node_in_rad);
 if (kepadatan < treshold && agent_->radius_
     < radius_maksimal && agent_->radius_ > 1
     && node_in_rad > 0){
     agent_->radius_ +=1;
 }

 if(kepadatan>treshold && agent_->radius_>1){
     agent_->radius_ -=1;
 }

 (agent_->iarpAgt_).buildRoutingTable();

 ...
}
```

**Kode Sumber 4.9** Pemilihan Radius dalam Kelas *IARPPeriodicTimer*

## 4.4 Implementasi Simulasi pada NS-2

Implementasi simulasi pada NS-2 dijalankan menggunakan kode program TCL. Dokumen ini berisi konfigurasi yang dibutuhkan untuk setiap *node* dan langkah-langkah yang dilakukan selama simulasi. Kode program untuk konfigurasi lingkungan simulasi dapat dilihat pada Kode Sumber 4.10.

```
#Channel Type
set val(chan) Channel/WirelessChannel;
#radio-propagationmodel
set val(prop) Propagation/TwoRayGround;
#network interface type
set val(netif) Phy/WirelessPhy;
#MAC type
set val(mac) Mac/802_11;
#interface queue type
set val(ifq) Queue/DropTail/PriQueue;
#link layer type
set val(ll)       LL;
#antenna model
set val(ant) Antenna/OmniAntenna;
#max packet in ifq
set val(ifqlen) 1000;
#number of mobilenodes
set val(nn) 100;
#Routing protocol
set val(rp) ZRP;
#Dimension of topography
set val(x) 1500;
set val(y) 1500;
#Simulation time
set val(stop) 200.0;
#Setting ZRP initial radius
Agent/ZRP set radius_ 1;
```

**Kode Sumber 4.10** Pengaturan Parameter Lingkungan Simulasi

Setelah melakukan konfigurasi lingkungan simulasi, langkah selanjutnya adalah melakukan inisialisasi program untuk skenario MANET. Kode program untuk inisialisasi skenario MANET dapat dilihat pada Kode Sumber 4.11.

```
#Create a ns simulator
set ns_ [new Simulator]

#Set up topography object
set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)

#Nam File Creation nam – network animator
set namtrace [open file_name.nam w]

#Tracing all the events and configuration
$ns_ namtrace-all $namtrace
$ns_ namtrace-all-wireless $namfile $val(x)
$val(y)

#Trace File creation
set tracefile [open file_name.tr w]

#Tracing all the events and cofiguration
$ns_ trace-all $tracefile

#general operational descriptor- storing the
hop details in the network
create-god $val(nn)

#Create wireless channel
set chan [new $val(chan)]
```

**Kode Sumber 4.11** Pengaturan Inisialisasi NS-2

Pada Kode Sumber 4.11, ns_ merupakan variabel baru untuk simulator. Pengaturan topografi juga dilakukan untuk menentukan

luas area yang akan digunakan saat simulasi, di mana dalam simulasi ini digunakan topografi dengan koordinat x dan y. Untuk menyimpan hasil simulasi, digunakan perintah *trace-all* di mana nantinya hasil simulasi akan dimasukkan ke dalam dokumen berekstensi .tr. Pada NS-2 juga disediakan *Network Animator* (biasa disebut NAM), yang memiliki sistem kerja hampir sama dengan dokumen .tr. Keduanya sama-sama melakukan *trace* terhadap skenario pada NS-2. Perbedaannya terletak pada hasil keluarannya, di mana NAM berbentuk visual dengan dokumen berekstensi .nam, sedangkan .tr berbentuk teks.

Sebelum membuat *node* untuk simulasi, hal pertama yang dilakukan adalah melakukan konfigurasi parameter *node* yang akan digunakan untuk simulasi. Konfigurasi *node* dapat terdiri dari tipe ad hoc *routing protocol*, *link layer*, MAC *layer*, dan lain sebagainya. Kode program untuk konfigurasi parameter *node* dapat dilihat pada Kode Sumber 4.12.

```
$ns_ node-config
      -adhocRouting $val(rp) \
      -llType $val(ll) \
      -macType $val(mac) \
      -ifqType $val(ifq) \
      -ifqLen $val(ifqlen) \
      -antType $val(ant) \
      -propType $val(prop) \
      -phyType $val(netif) \
      -topoInstance $topo \
      -agentTrace ON \
      -routerTrace ON \
      -macTrace ON \
      -movementTrace ON \
      -channel $chan
```

**Kode Sumber 4.12** Pengaturan Konfigurasi Parameter *Node*

Setelah melakukan konfigurasi parameter *node*, selanjutnya adalah membuat *node* dengan mendefinisikan *node-node* yang digunakan seperti menentukan posisi awal *node*, pergerakan *node*, dan pemberian label untuk masing-masing *node*. Pada tugas akhir ini, posisi *node* sumber dan tujuan diatur secara statis, sedangkan *node intermediate* diatur secara acak. Kode program untuk mendefinisikan *node* sumber dan tujuan dapat dilihat pada Kode Sumber 4.13.

```
#Node creation
for {set i 0} {$i < $val(nn)} {incr i} {
   set node_($i) [$ns_ node]
   $node_($i) random-motion 0
   $node_($i) color black
   $ns_ initial_node_pos $node_($i) 50
}

#set node source and destination
set val(source) [expr "$val(nn) - 2"]
set val(destination) [expr "$val(nn) - 1"]

#set source position
$node_($val(source)) set X_ 0.0
$node_($val(source)) set Y_ 750.0
$node_($val(source)) set Z_ 0.0

#set destination position
$node_($val(destination)) set X_ 1500.0
$node_($val(destination)) set Y_ 750.0
$node_($val(destination)) set Z_ 0.0
```

**Kode Sumber 4.13** Posisi Statis *Node* Sumber dan *Node* Tujuan

Pada Kode Sumber 4.13, dapat diketahui bahwa *node* sumber didefinisikan dengan jumlah *node* dikurangi 2, sedangkan untuk *node* tujuan adalah jumlah *node* dikurangi 1. Selanjutnya, pada Kode Sumber 4.13 juga dijelaskan bahwa posisi *node* sumber

mempunyai koordinat posisi yaitu 0, 750 (sumbu x, y) dan kedalaman 0 (sumbu z). Sedangkan untuk *node* tujuan mempunyai koordinat posisi yaitu 1500, 750 (sumbu x, y) dan kedalaman 0 (sumbu z).

Selanjutnya, penulis mengatur posisi *node intermediate* secara acak dengan aturan untuk sumbu x acak mulai dari 0 hingga 1500, sumbu y dibuat statis dengan variasi titik koordinat dari 0 hingga 1500, dan sumbu z tetap pada titik 0. Penentuan *node* secara acak tergantung pada jumlah *node* simulasinya. Kode program untuk mendefinisikan *node* sumber dan tujuan dapat dilihat pada Kode Sumber 4.14. Kode selengkapnya dapat dilihat pada lampiran 6.

```
set node 0
for {set i 0} {$i < 5} {incr i} {
   set distance 250
   $node_($node) set X_ [expr {($distance*$i)
                        + $distance}]
   $node_($node) set Y_ 750.0
   $node_($node) set Z_ 0.0
   $ns_ initial_node_pos $node_($node) 50
   set node [expr $node+1]
}

for {set i 0} {$i < 6} {incr i} {
   set distance 250
   $node_($node) set X_ [expr {($distance*$i)
                        + 125}]
   $node_($node) set Y_ 925.0
   $node_($node) set Z_ 0.0
   $ns_ initial_node_pos $node_($node) 50
   set node [expr $node+1]
}
```

**Kode Sumber 4.14** Posisi Acak *Node Intermediate*

Berikutnya adalah pemberian warna sebagai penanda untuk mempermudah pengecekan pada NAM saat melakukan simulasi. Pemberian warna *node* sesuai dengan peran dari masing-masing *node*. Warna hijau menyatakan bahwa *node* tersebut adalah *node* sumber, warna biru menyatakan *node* tujuan, dan warna hitam menyatakan *node intermediate*. Kode program untuk memberi label dan warna pada *node* sumber, tujuan, dan *intermediate* dapat dilihat pada Kode Sumber 4.15.

```
# Label and coloring
for {set i 0} {$i < $val(nn)-2} {incr i} {
  $ns_ at 0.0 "$node_($i) color black"
  $ns_ at 0.0 "$node_($i) label Node$i"
}

$node_($val(source)) color green
$ns_ at 0.0 "$node_($val(source)) color green"
$ns_ at 0.0 "$node_($val(source)) label
Source"

$node_($val(destination)) color blue
$ns_ at 0.0 "$node_($val(destination)) color
blue"
$ns_ at 0.0 "$node_($val(destination)) label
Destination"
```

**Kode Sumber 4.15** Pemberian Label dan Warna pada *Node*

Langkah selanjutnya adalah melakukan konfigurasi untuk mengatur pergerakan *node* pada simulasi. Dalam tugas akhir ini, pergerakan *node* dibuat secara acak. Pergerakan *node* secara acak menggunakan fitur *setdest* pada NS-2, dan menggunakan fitur *rand*() yang merupakan fungsi bawaan dari bahasa pemrograman C++. Waktu dimulainya perpindahan untuk setiap *node* juga diatur

secara acak yang dimulai dari detik ke-1. Kode program untuk mengatur pergerakan *node* dapat dilihat pada Kode Sumber 4.16.

```
$ns_ at 0.0 "destination"
proc destination {} {
   global ns_ val node_
   set time 1.0
   set now [$ns_ now]
   for {set i 0} {$i < $val(nn)} {incr i} {
       if {$i%3 == 0} {
           $ns_ at $now "$node_($i) setdest
                         [expr rand()*$val(x)]
                         [expr rand()*$val(y)]
                         5.0"
       }

       if {$i%3 == 1} {
           $ns_ at $now "$node_($i) setdest
                         [expr rand()*$val(x)]
                         [expr rand()*$val(y)]
                         5.0"
       }

       if {$i%3 == 2} {
           $ns_ at $now "$node_($i) setdest
                         [expr rand()*$val(x)]
                         [expr rand()*$val(y)]
                         5.0"
       }
   }
   $ns_ at [expr $now+$time] "destination"
}
```

**Kode Sumber 4.16** Konfigurasi Pergerakan *Node*

Setelah melakukan konfigurasi posisi, label, warna, dan pergerakan *node* untuk simulasi, langkah selanjutnya adalah mendefinisikan koneksi untuk simulasi, di mana pada tugas akhir

ini penulis menggunakan koneksi UDP. Dalam pengaturan koneksi UDP ini, ditentukan juga ukuran paket sebesar 512 *bytes* dengan interval waktu pengiriman sebesar 0,25 detik dan waktu simulasi yang dimulai pada detik ke 2,5 hingga detik ke 200. Kode program konfigurasi koneksi UDP dapat dilihat pada Kode Sumber 4.17.

```
#Setup a UDP connection
set udp0 [new Agent/UDP]
$ns_ attach-agent $node_($val(source)) $udp0
set null0 [new Agent/Null]
$ns_ attach-agent $node_($val(destination))
$null0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 512
$cbr0 set interval_ 0.25
$cbr0 set random_ 1
$cbr0 set maxpkts_ 10000
$cbr0 attach-agent $udp0
$ns_ connect $udp0 $null0
$ns_ at 2.5 "$cbr0 start"
$ns_ at [expr $val(stop)] "$cbr0 stop"
```

**Kode Sumber 4.17** Konfigurasi Koneksi Simulasi

Langkah selanjutnya adalah melakukan pengaturan untuk menyimpan hasil simulasi dalam bentuk .tr dan .nam. Kode program pengaturan untuk menyimpan hasil simulasi dapat dilihat pada Kode Sumber 4.18.

```
proc finish {} {
  global ns_ trfile namtrace
  $ns_ flush-trace
  close $namtrace
  exec nam file_name.nam &
  exit 0
}
```

**Kode Sumber 4.18** Konfigurasi Menyimpan Hasil Simulasi

Setelah melakukan konfigurasi menyimpan hasil simulasi, penulis melakukan konfigurasi untuk mengakhiri simulasi. Kode program untuk mengakhiri simulasi dapat dilihat pada Kode Sumber 4.19.

```
for {set i 0} {$i < $val(nn)} {incr i} {
  $ns_ at $val(stop) "$node_($i) reset";
}

$ns_ at $val(stop) "$ns_ nam-end-wireless
                    $val(stop)"
$ns_ at $val(stop) "stop"
$ns_ at $val(stop) "finish"

$ns_ at $val(stop) "puts \"NS EXITING...\";
                    $ns_ halt"

proc stop {} {
  global ns_ trfile
  $ns_ flush-trace
  close $trfile
}
```

**Kode Sumber 4.19** Konfigurasi Mengakhiri Simulasi

Implementasi simulasi pada dokumen TCL telah selesai. Untuk menjalankan skrip TCL, digunakan perintah yang dijalankan di terminal seperti pada Kode Sumber 4.20.

```
$ ns namafile.tcl
$ nam namafile.nam
```

**Kode Sumber 4.20** Perintah Menjalankan Simulasi TCL

Ketika skenario selesai dijalankan, maka akan dihasilkan dua dokumen, yaitu dokumen berekstensi .tr sebagai dokumen *trace route*, dan dokumen berekstensi .nam sebagai *network animator*.

## 4.5 Implementasi Metrik Analisis

Simulasi yang telah dijalankan oleh NS-2 menghasilkan keluaran berupa *trace file* yang berisikan data mengenai aktivitas yang terjadi selama simulasi dalam bentuk *plain text* berekstensi .tr. Dari data tersebut, dapat dilakukan analisis performa *routing protocol* ZRP dengan mengukur beberapa metrik. Pada tugas akhir ini, metrik yang akan dianalisis adalah *Packet Delivery Ratio* (PDR), *Routing Overhead* (RO), dan *Delivery Delay* (DD).

### 4.5.1 Implementasi *Packet Delivery Ratio*

Pada subbab 3.6.1, telah ditunjukkan contoh struktur data *event* dalam *trace file* oleh NS-2 dan telah dijelaskan cara menghitung PDR. Skrip AWK untuk menghitung PDR dapat dilihat pada lampiran 7.

Dalam menghitung PDR, kata kunci yang perlu diperhatikan pada *trace file* adalah *layer* AGT, karena kata kunci tersebut menunjukkan *event* yang bersangkutan dengan paket komunikasi data. Kemudian menghitung jumlah paket yang dikirimkan dan paket yang diterima dengan menggunakan karakter pada kolom pertama sebagai filter, karena kolom pertama yang menunjukkan *event* yang terjadi dari sebuah paket. Setelah itu, nilai PDR dapat dihitung dengan persamaan yang telah dijelaskan pada subbab 3.6.1. *Pseudocode* untuk menghitung PDR dapat dilihat pada Kode Sumber 4.21.

```
sent = 0
received = 0
for i = 1 to the number of rows
   if in a row contains "s" and AGT then
       sent++
   else if in a row contains "r" and AGT then
       eeceived++
   end if
pdr = received / sent
```

**Kode Sumber 4.21** *Pseudocode* Menghitung PDR

Penghitungan PDR pada tugas akhir ini dilakukan setelah simulasi dijalankan. Untuk menjalankan skrip AWK PDR menggunakan perintah awk -f pdr.awk *nama_trace_file*.tr.

### 4.5.2 Implementasi *Routing Overhead*

Pada subbab 3.6.2, telah ditunjukkan contoh struktur data *event* dalam *trace file* oleh NS-2 dan telah dijelaskan cara menghitung RO. Skrip AWK untuk menghitung RO dapat dilihat pada lampiran 8.

Seperti yang telah dijelaskan sebelumnya, RO merupakan jumlah dari *routing* paket kontrol baik itu paket RREQ, paket RREP, maupun paket RERR. Sehingga, untuk mendapatkan *route request* hal yang perlu dilakukan adalah menjumlahkan tiap paket dengan filter *event sent* pada kolom pertama dan *event layer* RTR pada kolom ke-4. *Pseudocode* untuk menghitung RO dapat dilihat pada Kode Sumber 4.22.

```
ro = 0
for i = 1 to the number of rows
     if in a row contains "s" or "f" and RTR
then
        ro++
     end if
```

**Kode Sumber 4.22** *Pseudocode* Menghitung RO

Pada tugas akhir ini, penghitungan RO juga dilakukan setelah simulasi dijalankan Untuk menjalankan skrip awk RO menggunakan perintah awk -f ro.awk *nama_trace_file*.tr.

### 4.5.3 Implementasi *Delivery Delay*

Pada subbab 3.6.3, telah ditunjukkan contoh struktur data *event* dalam *trace file* oleh NS-2 dan telah dijelaskan cara menghitung *Delivery Delay*. Skrip AWK untuk menghitung *Delivery Delay* dapat dilihat pada lampiran 9.

Dalam menghitung *Delivery Delay*, kata kunci yang perlu diperhatikan pada *trace file* adalah *layer* AGT, karena kata kunci tersebut menyatakan bahwa aktivitas terjadi pada layer *agent* atau aplikasi di mana pengiriman sudah berupa paket asli. Selain itu, untuk mengetahui waktu yang dibutuhkan selama pengiriman paket data dapat dilihat pada kolom kedua *trace file,* dengan merujuk pada kolom pertama filter *event* di mana kata kunci *sent* (s) menandakan pengiriman paket dan *received* (r) yang berarti penerimaan paket. Setelah itu nilai *Delay Delivery* dapat dihitung dengan persamaan yang telah dijelaskan pada subbab 3.6.3. *Pseudocode* untuk menghitung *Delivery Delay* dapat dilihat pada Kode Sumber 4.23.

```
time_sent = 0
time_received = 0
for i=1 to the number of rows
      if in a row contains “s” and AGT
then
            time_sent++
      else in a row contains “r” and AGT
then
            time_received++
      end if
dd = time_received - time_sent
```

**Kode Sumber 4.23** *Pseudocode* Menghitung *Delivery Delay*

Pada tugas akhir ini, penghitungan *Delivery Delay* juga dilakukan setelah simulasi dijalankan. Untuk menjalankan skrip AWK *Delivery Delay* menggunakan perintah awk -f dd.awk *nama-trace-file*.tr.

***(Halaman ini sengaja dikosongkan)***

# BAB V
# UJI COBA DAN EVALUASI

Bab ini membahas mengenai uji coba yang dilakukan dan evaluasi sesuai dengan rancangan dan implementasi. Dari hasil yang didapatkan setelah melakukan uji coba, akan dilakukan evaluasi sehingga dapat diambil kesimpulan untuk bab selanjutnya.

## 5.1 Lingkungan Uji Coba

Uji coba dilakukan pada perangkat dengan spesifikasi seperti yang tertera pada Tabel 5.1.

**Tabel 5.1** Spesifikasi Perangkat yang Digunakan

| Komponen | Spesifikasi |
|---|---|
| **CPU** | Intel(R) Core(TM) i5-7400 CPU @ 3.00GHz 3.00 GHz |
| **Sistem Operasi** | Ubuntu Bionic 18.04 LTS 64 bit |
| **Linux Kernel** | Linux kernel 4.4 |
| **Memori** | RAM 8 GB |
| **Penyimpanan** | 928 GB |

Pengujian dilakukan dengan menjalankan skenario yang disimulasikan menggunakan NS-2. Dari simulasi tersebut dihasilkan sebuah *trace file* dengan ekstensi .tr yang akan dianalisis menggunakan skrip AWK untuk mendapatkan *Packet Delivery Ratio*, *Routing Overhead*, dan *Delivery Delay* dengan kode masing-masing pada lampiran 7. Kode Skrip AWK *Packet Delivery Ratio*, lampiran 8. Kode Skrip AWK *Routing Overhead*, dan lampiran 9. Kode Skrip AWK *Delivery Delay*.

## 5.2 Hasil Uji Coba

Simulasi yang telah dijalankan oleh NS-2 hingga proses analisis menggunakan skrip AWK menghasilkan keluaran berupa nilai metrik analisis. Sebelumnya, pada subbab 3.3 telah dibahas mengenai skenario penentuan *threshold* yang menggunakan

*routing protocol* ZRP modifikasi. Pada bagian ini, akan dibahas mengenai hasil pra-uji coba penentuan *threshold* dan hasil uji coba skenario simulasi NS-2 menggunakan *routing protocol* ZRP yang telah dimodifikasi.

### 5.2.1 Hasil Pra-Uji Coba Penentuan *Threshold*

Sebelum memasuki hasil uji coba skenario simulasi NS-2, dilakukan terlebih dahulu pra-uji coba penentuan nilai *threshold* sesuai dengan skenario yang telah dirancang pada subbab 3.3. Dalam penentuan *threshold*, parameter lingkungan simulasi diatur sama dengan parameter lingkungan yang telah dijelaskan dalam subbab 3.5. Perbedaannya terletak pada jumlah dan posisi *node*, di mana dalam pra-uji coba penentuan *threshold* ini jumlah *node* yang digunakan adalah 25 *node* serta posisi *node* dibuat statis dan tidak bergerak. Selain itu, pada penentuan *threshold* ini menggunakan *routing protocol* ZRP yang telah dimodifikasi.

Penentuan *threshold* dilakukan dengan menggunakan besar kepadatan *node* tetangga dari 0 sampai batas tertentu, yaitu ketika nilai *Packet Delivery Ratio, Routing Overhead,* dan *Delivery Delay* menghasilkan angka yang sama untuk kepadatan yang berbeda. Simulasi dilakukan satu kali untuk setiap besaran *threshold*. Hal itu dilakukan karena posisi dan pergerakan *node* dalam pra-uji coba ini dibuat statis dan tidak bergerak, sehingga akan menghasilkan nilai metrik analisis yang sama pada besar kepadatan *node* tetangga yang sama.

Dari simulasi pra-uji coba yang telah dilakukan, hasil metrik analisis menunjukkan bahwa pada saat kepadatan *node* tetangga sebesar 0,52, nilai PDR mencapai angka 1. Begitupun dengan metrik analisis untuk kepadatan *node* tetangga hingga 1,00 menunjukkan hasil metrik analisis yang sama untuk nilai PDR, RO, dan DD. Sehingga, hasil pra-uji coba penentuan *threshold* didapatkan angka *threshold* sebesar 0,52. Hasil pra-uji coba penentuan *threshold* selengkapnya dapat dilihat pada Tabel 5.2.

**Tabel 5.2** Hasil Pra-Uji Coba Penentuan *Threshold*

| ***Density*** | ***Packet Delivery Ratio*** | ***Routing Overhead*** | ***Delivery Delay*** |
|---|---|---|---|
| 0,04 | 0,9975 | 11.613 | 83,5084 |
| 0,08 | 0,9975 | 12.721 | 88,3990 |
| 0,12 | 0,6013 | 10.117 | 194,1120 |
| 0,16 | 0,6146 | 10.243 | 44,4640 |
| 0,20 | 0,6184 | 10.436 | 41,9038 |
| 0,24 | 0,6247 | 10.766 | 41,9407 |
| 0,28 | 0,1043 | 8.449 | 33,5361 |
| 0,32 | 0,6038 | 12.019 | 509,8210 |
| 0,36 | 0,6090 | 12.290 | 52,8119 |
| 0,40 | 0,6112 | 12.529 | 498,0540 |
| 0,44 | 0,3654 | 11.294 | 40,0569 |
| 0,48 | 0,6197 | 13.065 | 42,3226 |
| 0,52 | 1,0000 | 15.052 | 47,6143 |

Dari hasil pra-uji coba penentuan *threshold*, dapat diketahui bahwa ketika *density* atau kepadatan *node* tetangga sebesar 0,52, nilai *Packet Delivery Ratio* menunjukkan angka 1, di mana angka tersebut merupakan nilai tertinggi dari setiap percobaan pada besar kepadatan *node* tetangga yang dilakukan. Sehingga, pada tugas akhir ini 0,52 dipilih sebagai nilai *threshold* kepadatan *node* tetangga. Selanjutnya, nilai *threshold* tersebut digunakan dalam simulasi NS-2 untuk mengetahui performa metrik analisisnya.

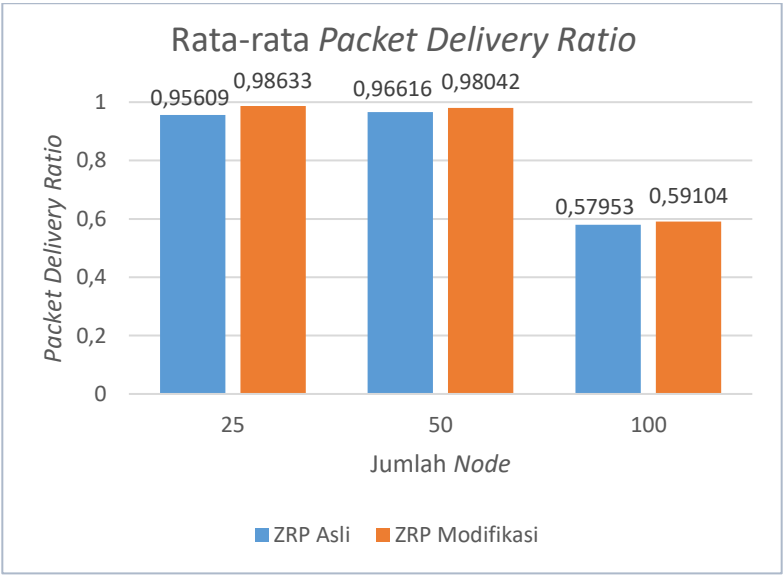
### 5.2.2 Hasil Uji Coba Simulasi NS-2

Pengujian pada skenario simulasi NS-2 dilakukan untuk melihat perbandingan metrik analisis berupa *Packet Delivery Ratio*, *Routing Overhead*, dan *Delivery Delay* antara *routing protocol* ZRP asli dan ZRP yang telah dimodifikasi. Pengujian

dilakukan sesuai dengan perancangan skenario yang telah dijelaskan pada subbab 3.5

Pengambilan data uji metrik analisis dengan luas area 1500 m x 1500 m dan *node* sebanyak 25, 50, dan 100 menggunakan nilai *threshold* yang telah didapatkan pada hasil pra-uji coba yaitu sebesar 0,52. Nilai *threshold* ini bermakna bahwa maksimal *node* di dalam zona IARP yang dimiliki oleh suatu *node* tidak boleh melebihi nilai *threshold*, dan nilai radius minimal adalah 1. Uji coba simulasi NS-2 dilakukan sebanyak sepuluh kali, kemudian dihitung nilai rata-rata untuk setiap parameter metrik analisis yang telah ditentukan.

Hasil pengambilan data metrik analisis berupa *Packet Delivery Ratio* pada *node* 25, 50, dan 100 dengan menggunakan *routing protocol* ZRP asli dan ZRP yang telah dimodifikasi dapat dilihat pada Gambar 5.1.

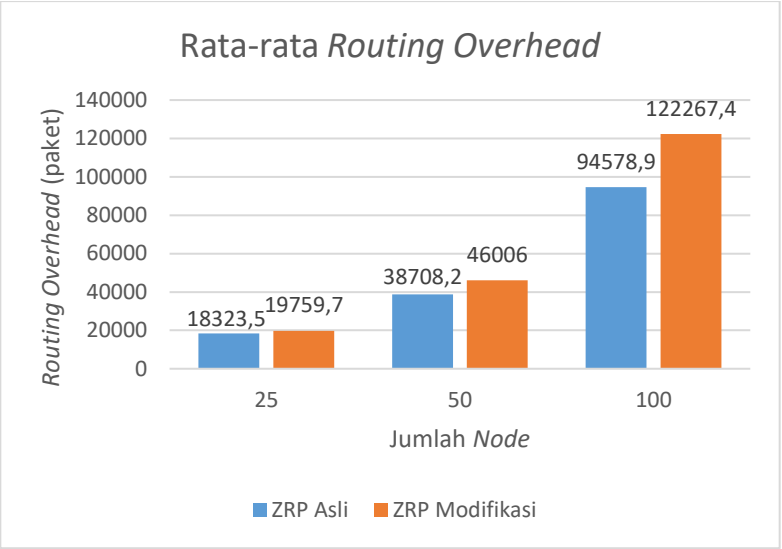



**Gambar 5.1** Grafik Rata-rata *Packet Delivery Ratio* pada Simulasi NS-2

Berdasarkan grafik pada Gambar 5.1, dapat dilihat bahwa *routing protocol* ZRP asli dan ZRP yang telah dimodifikasi mengalami kenaikan nilai *Packet Delivery Ratio*. Pada lingkungan dengan jumlah *node* 25, menghasilkan perbedaan selisih PDR sebesar 0,03024, atau naik sekitar 3,16 % di mana *routing protocol* ZRP yang telah dimodifikasi unggul dalam nilai PDR tersebut. Pada lingkungan dengan jumlah *node* 50, menghasilkan perbedaan selisih PDR sebesar 0,01426, atau naik sekitar 1,48 % di mana *routing protocol* ZRP yang telah dimodifikasi juga unggul dalam nilai PDR tersebut. Pada lingkungan dengan jumlah *node* 100, menghasilkan perbedaan selisih PDR sebesar 0,01151, atau naik sekitar 1,99 % di mana *routing protocol* ZRP yang telah dimodifikasi juga unggul dalam nilai PDR tersebut.

Dapat dilihat rata-rata kenaikan yang terjadi untuk nilai PDR adalah 2,21 %. Jika ketiga lingkungan variasi jumlah *node* tersebut dibandingkan, dapat dilihat bahwa dengan jumlah *node* 25 menghasilkan PDR yang lebih bagus daripada di lingkungan dengan jumlah *node* yang lebih banyak untuk *routing protocol* ZRP yang telah dimodifikasi. Sedangkan pada ZRP asli, nilai PDR yang lebih bagus dihasilkan oleh jumlah *node* 50. Dapat dilihat pula bahwa ZRP yang telah dimodifikasi menghasilkan PDR yang lebih bagus atau dalam hal ini lebih tinggi daripada *routing protocol* ZRP asli.

Hasil pengambilan data metrik analisis berupa *Routing Overhead* pada *node* 25, 50, dan 100 dengan menggunakan *routing protocol* ZRP asli dan ZRP yang telah dimodifikasi dapat dilihat pada Gambar 5.2.



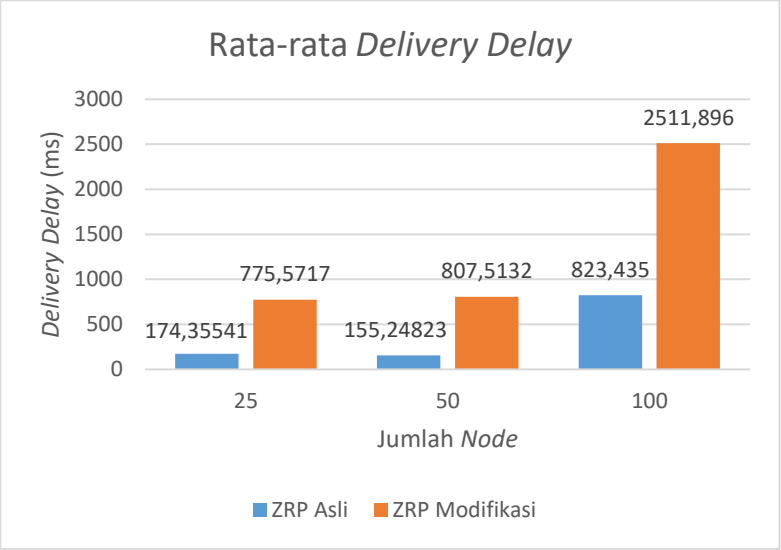

**Gambar 5.2** Grafik Rata-rata *Routing Overhead* pada Simulasi NS-2

Berdasarkan grafik pada Gambar 5.2, dapat dilihat bahwa *routing protocol* ZRP asli dan ZRP yang telah dimodifikasi mengalami kenaikan nilai *Routing Overhead*. Pada lingkungan dengan jumlah *node* 25, menghasilkan perbedaan selisih RO sebesar 1436,2, atau naik sekitar 7,84 % di mana *routing protocol* ZRP yang telah dimodifikasi lebih unggul dalam nilai RO tersebut. Pada lingkungan dengan jumlah *node* 50, menghasilkan perbedaan selisih RO sebesar 7297,8, atau naik sekitar 18,85 % di mana *routing protocol* ZRP yang telah dimodifikasi juga unggul dalam nilai RO tersebut. Pada lingkungan dengan jumlah *node* 100, menghasilkan perbedaan selisih RO sebesar 27688,5, atau naik sekitar 29,28 % di mana *routing protocol* ZRP yang telah dimodifikasi juga unggul dalam nilai RO tersebut.

Dapat dilihat rata-rata kenaikan yang terjadi untuk nilai RO adalah 18,66 %. Jika ketiga lingkungan variasi jumlah *node* tersebut dibandingkan, dapat dilihat bahwa dengan jumlah *node* 25

menghasilkan RO yang lebih bagus atau lebih sedikit daripada lingkungan dengan jumlah *node* yang lebih banyak untuk *routing protocol* ZRP asli maupun yang telah dimodifikasi.

Hasil pengambilan data metrik analisis berupa *Delivery Delay* pada *node* 25, 50, dan 100 dengan menggunakan *routing protocol* ZRP asli dan ZRP yang telah dimodifikasi dapat dilihat pada Gambar 5.3.




**Gambar 5.3** Grafik Rata-rata *Delivery Delay* pada Simulasi NS2

Berdasarkan grafik pada Gambar 5.3, dapat dilihat bahwa *routing protocol* ZRP asli dan ZRP yang telah dimodifikasi mengalami kenaikan nilai *Delivery Delay*. Pada lingkungan dengan jumlah *node* 25, menghasilkan perbedaan selisih *Delivery Delay* sebesar 601,22, atau naik sekitar 345 % di mana *routing protocol* ZRP yang telah dimodifikasi lebih unggul dalam nilai *Delivery Delay* tersebut. Pada lingkungan dengan jumlah *node* 50, menghasilkan perbedaan selisih *Delivery Delay* sebesar 652,26, atau naik sekitar 420 % di mana *routing protocol* ZRP yang telah

dimodifikasi juga unggul dalam nilai *Delivery Delay* tersebut. Pada lingkungan dengan jumlah *node* 100, menghasilkan perbedaan selisih *Delivery Delay* sebesar 1688,46, atau naik sekitar 205 % di mana *routing protocol* ZRP yang telah dimodifikasi juga unggul dalam nilai *Delivery Delay* tersebut.

Dapat dilihat rata-rata kenaikan yang terjadi untuk nilai *Delivery Delay* adalah 323 %. Jika ketiga lingkungan variasi jumlah *node* tersebut dibandingkan, dapat dilihat bahwa dengan jumlah *node* 25 menghasilkan *Delivery Delay* yang lebih bagus atau lebih sedikit daripada lingkungan dengan jumlah *node* yang lebih banyak untuk *routing protocol* ZRP yang telah dimodifikasi. Sedangkan untuk *routing protocol* ZRP asli, nilai *Delivery Delay* yang lebih bagus dihasilkan oleh *node* 50.

# BAB VI
# KESIMPULAN DAN SARAN

Bab ini membahas mengenai kesimpulan yang diperoleh dari tugas akhir yang telah dikerjakan dan saran terkait pengembangan dari tugas akhir ini yang dapat dilakukan pada masa yang akan datang.

## 6.1 Kesimpulan

Kesimpulan yang diperoleh dari hasil uji coba dan evaluasi pada tugas akhir ini adalah sebagai berikut:

1. Dengan menambahkan penghitungan kepadatan *node* tetangga dan membandingkannya dengan nilai *threshold*, maka radius yang adaptif terhadap kepadatan *node* tetangga dapat diimplementasikan.
2. Pengaruh zonasi yang adaptif terhadap performa *routing protocol* ZRP secara keseluruhan untuk skenario simulasi NS-2 menghasilkan peningkatan rata-rata *Packet Delivery Ratio* sebesar 2,21 %, peningkatan rata-rata *Routing Overhead* sebesar 18,66%, dan peningkatan rata-rata *Delivery Delay* sebesar 323%.

## 6.2 Saran

Saran yang diberikan dari hasil uji coba dan evaluasi pada tugas akhir ini adalah sebagai berikut:

1. Untuk mendapatkan hasil uji coba yang lebih baik, dapat dilakukan uji coba yang lebih banyak, misalnya sepuluh kali.
2. Untuk mendapatkan hasil uji coba yang lebih baik, dapat dilakukan penentuan posisi dan pergerakan *node* yang lebih teratur atau dengan kata lain semi-acak.
3. Ide untuk implementasi zonasi yang adaptif terhadap kepadatan *node* tetangga sudah bagus, kedepannya bisa mengimplementasikan dengan menambahkan aspek lain yang dijadikan untuk parameter zonasi yang adaptif seperti kecepatan, arah, dan lain sebagainya.

***(Halaman ini sengaja dikosongkan)***

# DAFTAR PUSTAKA


[1] S. Adiwicaksono, "DETEKSI MALICIOUS *NODE* PADA ZONE *ROUTING*," Teknik Informatika, Institut Teknologi Sepuluh Nopember, Surabaya, 2017.

[2] M. F. Rahman, K. S. N. Ripon, S. M. M. Karim and M. I. H. Suvo, "Performance Analysis of Estimation of Distribution," *International Journal of Computer Science and Information Security*, vol. 8, August 2010.

[3] A. Nasipuri, "Mobile Ad Hoc Networks," The University of North Carolina, Charlotte.

[4] N. Beijar, "Zone *Routing* Protocol (ZRP)," Finland, 2015.

[5] J. Sasongko, "Network Simulator dan Network Animator Menggunakan Cygnus Windows dalam Windows XP," vol. XIV, Januari 2009.

[6] Y. Joshi and D. Parekh, 2016. [Online]. Available: https://academic.csuohio.edu/yuc/mobile/mcproj/5d-MC%20Final%20Report%20.pdf. [Accessed 19 October 2018].

[7] P. R. L. D. Neha Singh and V. Mathur, "Network Simulator NS2-2.35," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 2, May 2012.

[8] A. V. Aho, B. W. Kerninghan and P. J. Weinberger, The AWK Programming Language, Boston: Addison-Wesley, 1988.

[9] Surateno, "OPTIMASI PENENTUAN ZONA PADA *ROUTING PROTOCOL* HOPNET DENGAN TEKNIK MIN-SEARCHING," Pasca Sarjana Teknik Informatika, Intitut Teknologi Sepuluh Nopember, Surabaya, 2011.

***(Halaman ini sengaja dikosongkan)***

## LAMPIRAN

### 1. Kode Konfigurasi Kelas *NDPAgent*

```
class NDPAgent{
  public:
    ZRPAgent *agent_;
    NeighborList neighborLst_;
    NDPBeaconTransmitTimer
      BeaconTransmitTimer_;
    NDPAckTimer AckTimer_;
    int startup_jitter_;
    NDPAgent(ZRPAgent *agent) :
      agent_(agent),
      BeaconTransmitTimer_(agent),
      AckTimer_(agent),
      startup_jitter_(DEFAULT_STARTUP_JITT
      ER) {}
    void startUp();
    void recv_NDP_BEACON(Packet* p);
    void recv_NDP_BEACON_ACK(Packet* p);
    int print_tables();
}
```

**2. Kode Konfigurasi Kelas *IARPAgent***

```
class IARPAgent{
  public:
    ZRPAgent *agent_;
    int updateSendFlag_;
    LinkStateList lsLst_;
    Peripheral_Node_List pnLst_;
    InnerRouteList irLst_;
    IARPUpdateDetectedList upLst_;
    int linkLifeTime_;
    int updateLifeTime_;

    IARPPeriodicUpdateTimer
      PeriodicUpdateTimer_;
    IARPExpirationTimer ExpirationTimer_;
    int startup_jitter_;

    IARPAgent(ZRPAgent *agent) :
      agent_(agent), updateSendFlag_(TRUE),
      linkLifeTime_(DEFAULT_LINK_LIFETIME),
      updateLifeTime_(DEFAULT_UPDATE_LIFETIME)
      ,PeriodicUpdateTimer_(agent),ExpirationT
      imer_(agent),startup_jitter_(DEFAULT_STA
      RTUP_JITTER){}
    void startUp();
    void build_Routing_Table();
    void addRouteInPacket(nsaddr_t dest,Packet
      *p);
    void recv_IARP_UPDATE(Packet* p);
    void recv_IARP_DATA(Packet* p);
    int count_node_in_rad();
};
```

**3. Kode Konfigurasi *constant.h***

```
#ifndef _constants_h_
#define _constants_h_

#define DEBUG    1
#define ROUTER_PORT   0xff
#define TRUE     1
#define FALSE    0
#define LINKUP   1
#define LINKDOWN 0
#define DEFAULT_STARTUP_JITTER 1
#define DEFAULT_EXPIRATION_CHECK_PERIOD 1
#define MAX_SEQUENCE_ID 1000000
#define MIN_PACKET_DROP_TIME 3
#define MAX_PACKET_DROP_TIME 10
#define IERP_TTL  50

#define DEFAULT_BEACON_PERIOD          3
#define DEFAULT_BEACON_PERIOD_JITTER   1
#define DEFAULT_NEIGHBOR_ACK_TIMEOUT   2
#define DEFAULT_MAX_ACK_TIMEOUT        2

#define DEFAULT_MIN_IARP_UPDATE_PERIOD 3
#define DEFAULT_MAX_IARP_UPDATE_PERIOD 9
#define DEFAULT_LINK_LIFETIME          10
#define DEFAULT_UPDATE_LIFETIME  30

#define DEFAULT_BRP_XMIT_POLICY        0
#define IERP_REPLY_SNOOP          1
#define IERP_ERROR_SNOOP          1
#define IERP_XMIT_JITTER          1
#define DEFAULT_QUERY_LIFETIME    30
#define DEFAULT_QUERY_RETRY_TIME 5
#define DEFAULT_ROUTE_LIFETIME    120
#define DEFAULT_MAX_IERP_REPLY    3
```

```
#define ZRP_DEFAULT_HDR_LEN            10
#define DEFAULT_ZONE_RADIUS            2
#define DEFAULT_SEND_BUFFER_SIZE       100
#define DEFAULT_PACKET_LIFETIME        20
#define DEFAULT_INTERPKT_JITTER        1

typedef double Time;
typedef int32_t Query_ID;

enum ZRPTYPE {
  NDP_BEACON, NDP_BEACON_ACK,
  IARP_UPDATE, IARP_DATA, IERP_REPLY,
  IERP_REQUEST, IERP_ROUTE_ERROR,
  IERP_DATA
};

enum BRP_XMIT_POLICY {
  BRP_UNICAST, BRP_MULTICAST
};

#endif
```

**4. Kode Fungsi NDPAckTimer::handle(Event* e)**

```
void NDPAckTimer::handle(Event* e) {
   if((agent_->ndpAgt_) .neighborLst_
   .isEmpty()){
      if(DEBUG) {
         Time now = Scheduler::instance()
         .clock();
         printf("\n_%d_ [%6.6f] |
          NDP_ISOLATED_NODE | No NDP_BEACON_ACK
          Detected | ",agent_->myaddr_, now);
          agent_->print_tables(); printf("\n");
      }return;
   }
   Time now = Scheduler::instance() .clock();
   Neighbor *prev, *curNb;
   prev = NULL;
   curNb = (agent_->ndpAgt_).neighborLst_
    .head_;
   for(; curNb!=NULL; ) {
      if((now - curNb->lastack_) >
      neighbor_ack_timeout_) {
         curNb->AckTOCount_++;
            if(curNb->AckTOCount_ >=
            DEFAULT_MAX_ACK_TIMEOUT) {
               (agent_->iarpAgt_).
                updateSendFlag_ = TRUE;
               Linkstate *handleToFoundLS = NULL;
               int foundFlag;
               foundFlag = (agent_-
                >iarpAgt_).lsLst_.findLink(curNb-
                >addr_, agent_->myaddr_,
                &handleToFoundLS);
               if( foundFlag == TRUE ) {
                   handleToFoundLS->isup_ =
                   LINKDOWN;
                   (agent_->iarpAgt_)
                   .build_Routing_Table();}
```

```
            curNb->linkStatus_ = LINKDOWN;
         }

         if(DEBUG) {
            Time now = Scheduler::instance()
            .clock();
            printf("\n_%d_ [%6.6f] |
            NDP_BEACON_ACK_TIMEOUT | -S %d -Nb
            %d | -TimeOut %d | ",agent_-
            >myaddr_, now, agent_->myaddr_,
            curNb->addr_,
            neighbor_ack_timeout_);
            agent_->print_tables();
            printf("\n");
         }
      }
      prev = curNb;
      curNb=curNb->next_;
}

   if (agent_->radius_==1) {
      int num_neighbor = (agent_-> ndpAgt_)
                          .print_tables();
      float density = (float)num_neighbor /
                         jumlah_node_simulasi;
      float th = nilai_threshold;
      printf("\nNDP UPDATED FOR NODE NUMBER :
      %d, radius : %d, kepadatan : %f,
      num_neighbor : %d\n\n", agent_->myaddr_,
      agent_->radius_, density, num_neighbor);

      if (density < th && num_neighbor > 0){
         agent_->radius_ +=1;
      }
   }
}
```

**5. Kode Fungsi IARPPeriodicUpdateTimer::handle (Event* e)**

```
void IARPPeriodicUpdateTimer::handle(Event*
e){
   if(agent_->radius_ == 1) {
      Scheduler::instance().schedule(this,
      &intr_, min_iarp_update_period_);
      return;
   }
   Time now = Scheduler::instance().clock();
   if((agent_->iarpAgt_) .updateSendFlag_
   ==FALSE && (now-lastUpdateSent_)
   <max_iarp_update_period_) {
      if(DEBUG) {
         Time now = Scheduler::instance()
         .clock();
         printf("\n_%d_ [%6.6f] |
         MISS_IARP_UPDATE | No Neighbor Changes
         | -Tlsu %d | ",agent_->myaddr_, now,
         min_iarp_update_period_);
         agent_->print_tables();
         printf("\n");
      }
      Scheduler::instance().schedule(this,
      &intr_, min_iarp_update_period_);
      return;
   }

   if((agent_->ndpAgt_).neighborLst_.isEmpty()
   == FALSE) {
      Packet* p;
      p = (agent_->pktUtil_). pkt_create
      (IARP_UPDATE, IP_BROADCAST, agent_-
      >radius_-1);
      (agent_->pktUtil_).pkt_add_LSU_space(p,
      (agent_->ndpAgt_) .neighborLst_
      .numNeighbors_);
```

```
hdr_zrp *hdrz = HDR_ZRP(p);
Neighbor *cur = (agent_->ndpAgt_).
neighborLst_.head_;
for(int i=0; i<(agent_->ndpAgt_)
.neighborLst_.numNeighbors_; i++) {
  hdrz->links_[i].src_ = agent_-
  >myaddr_;
  hdrz->links_[i].dest_ = cur->addr_;
  hdrz->links_[i].isUp_ = cur-
  >linkStatus_;
  cur = cur->next_;}
hdrz->numlinks_ = (agent_->ndpAgt_).
neighborLst_.numNeighbors_;
(agent_->pktUtil_).pkt_broadcast(p,
0.00);
if(DEBUG) {
  Time now = Scheduler::instance()
  .clock();
  hdr_zrp *hdrz = HDR_ZRP(p);
  hdr_ip *hdrip = HDR_IP(p);
  printf("\n_%d_ [%6.6f] |
  XMIT_IARP_UPDATE | -S %d -IS %d | -
  SEQ %d | ",agent_->myaddr_, now,
  hdrz->src_, hdrip->saddr(), hdrz-
  >seq_);
  (agent_->pktUtil_)
  .pkt_print_links(p);
  agent_->print_tables();
  printf("\n");}
(agent_->iarpAgt_).upLst_.addUpdate
(hdrz->src_, hdrz->seq_,now+(agent_-
>iarpAgt_).updateLifeTime_);
(agent_->iarpAgt_).updateSendFlag_ =
FALSE;
lastUpdateSent_ = now;
(agent_->ndpAgt_).neighborLst_
.purgeDownNeighbors();}
```

```
else{
 if(DEBUG) {
  Time now = Scheduler::instance().clock();
  printf("\n_%d_ [%6.6f] | IARP_ISOLATED_NODE
  | No Neighbors | ",agent_->myaddr_, now);
  agent_->print_tables(); printf("\n");
 }
}

int node_in_rad = (agent_-> iarpAgt_)
.count_node_in_rad()-1;

float kepadatan = (float)node_in_rad /
jumlah_node_simulasi;

float treshold = nilai_threshold;

printf("\nIARP UPDATED FOR NODE NUMBER : %d,
       radius : %d, kepadatan : %f,
       node_in_rad : %d\n\n", agent_->myaddr_,
       agent_->radius_, kepadatan,
       node_in_rad);
if(kepadatan < treshold && agent_->radius_ <
  radius_maksimal && agent_->radius_ > 1 &&
  node_in_rad > 0){
    agent_->radius_ +=1;
}

if(kepadatan>treshold && agent_->radius_>1){
  agent_->radius_ -=1;
}

(agent_->iarpAgt_).buildRoutingTable();

Scheduler::instance().schedule(this, &intr_,
min_iarp_update_period_);

}
```

**6. Kode Konfigurasi Posisi *Node* pada Simulasi NS-2**

```
set node 0
for {set i 0} {$i < 5 } { incr i } {
  set distance 250
  $node_($node) set X_ [expr {($distance*$i)
+ $distance}]
  $node_($node) set Y_ 750.0
  $node_($node) set Z_ 0.0
  $ns_ initial_node_pos $node_($node) 50
  set node [expr $node+1]
}
for {set i 0} {$i < 6 } { incr i } {
  set distance 250
  $node_($node) set X_ [expr {($distance*$i)
+ 125}]
  $node_($node) set Y_ 925.0
  $node_($node) set Z_ 0.0
  $ns_ initial_node_pos $node_($node) 50
  set node [expr $node+1]
}
for {set i 0} {$i < 6 } { incr i } {
  set distance 250
  $node_($node) set X_ [expr {($distance*$i)
+ 125}]
  $node_($node) set Y_ 525.0
  $node_($node) set Z_ 0.0
  $ns_ initial_node_pos $node_($node) 50
  set node [expr $node+1]
}
for {set i 0} {$i < 3 } { incr i } {
  set distance 250
  $node_($node) set X_ [expr {($distance*$i)
+ 500}]
  $node_($node) set Y_ 1150.0
  $node_($node) set Z_ 0.0
  $ns_ initial_node_pos $node_($node) 50
  set node [expr $node+1]
}
```

```
for {set i 0} {$i < 3 } { incr i } {
  set distance 250
  $node_($node) set X_ [expr {($distance*$i) +
500}]
  $node_($node) set Y_ 350.0
  $node_($node) set Z_ 0.0
  $ns_ initial_node_pos $node_($node) 50
  set node [expr $node+1]
}
```

**7. Kode Skrip AWK *Packet Delivery Ratio***

```
BEGIN {
     sendLine = 0;
     recvLine = 0;
     fowardLine = 0;
}

$0 ~/^s.* AGT/ {
     sendLine ++ ;
}

$0 ~/^r.* AGT/ {
     recvLine ++ ;
}

$0 ~/^f.* RTR/ {
     fowardLine ++ ;
}

END {
     printf "cbr s:%d r:%d, r/s
     Ratio:%.4f, f:%d \n", sendLine,
     recvLine,
     (recvLine/sendLine),fowardLine;
}
```

**8. Kode Skrip AWK *Routing Overhead***

```
BEGIN {
     rt_pkts = 0;
}
{
     if ($2 >= 101) {
          if (($1 == "s" || $1 == "f") &&
          ($4 == "RTR") && ($7 == "DSR"))
          {
               rt_pkts++;
          }
     }
}
END {
     printf "Routing Packets \t= %d \n",
     rt_pkts;
}
```

**9. Kode Skrip AWK *Delivery Delay***

```
BEGIN{
   sum_delay = 0; count = 0;
}
{
   if ($2 >= 101) {
      if($4=="AGT" && $1=="s" && seqno < $6) {
         seqno = $6;
      }
      if($4 == "AGT" && $1 == "s") {
         start_time[$6] = $2;
      }
      else if(($7 == "cbr") && ($1 == "r")) {
         end_time[$6] = $2;
      }
      else if($1 == "D" && $7 == "cbr") {
         end_time[$6] = -1;
      }
   }
}
END {
   for(i=0; i<=seqno; i++) {
      if(end_time[i] > 0) {
         delay[i] = end_time[i]-start_time[i];
         count++;}
      else {
         delay[i] = -1;}
   }
   for(i=0; i<=seqno; i++) {
      if(delay[i] > 0) {
         n_to_n_delay=n_to_n_delay+delay[i];
      }
   }
   n_to_n_delay = n_to_n_delay/count;
   printf "End-to-End Delay \t=
   "n_to_n_delay * 1000 " ms \n";
}
```

# BIODATA PENULIS

Hania Maghfira lahir di Pasuruan pada tanggal 19 Agustus 1997. Penulis menempuh pendidikan formal di TK Bustanul Athfal Purwokerto (2001-2003), SDN Dukuwaluh II Purwokerto (2003-2004), SDN Kebagusan 04 Pg. Jakarta Selatan (2004-2006), SDN Purutrejo II Pasuruan (2006-2007), MI Nurul Huda Gondol Buleleng Bali (2007-2009), MTsN PATAS Buleleng Bali (2009-2012), SMAN 2 Pasuruan (2012-2015), dan Informatika ITS Surabaya (2015-2019). Bidang studi yang diambil oleh penulis saat berkuliah di Departemen Informatika ITS adalah Arsitektur Jaringan Komputer (AJK). Penulis aktif dalam organisasi Badan Eksekutif Mahasiswa Fakultas Teknologi Informasi dan Komunikasi sebagai Wakil Ketua BEM periode 2017-2018, Himpunan Mahasiswa Teknik Computer-Informatika (2016-2017), dan Jamaah Masjid Manarul Ilmi (2016-2018). Penulis juga aktif dalam kegiatan kepanitian seperti SCHEMATICS 2016 - 2017 Divisi Humas, FTIF FESTIVAL 2017 sebagai Bendahara, dan INTEGRALISTIK FESTIVAL 2016 sebagai Sekretaris. Penulis pernah menjalani kerja praktik di PT PLN (Persero) Kantor Pusat Jakarta periode Januari 2018, magang di Kementerian Luar Negeri RI periode Juli-Agustus 2017 dan PT Toyota Astra Motor Jakarta periode Juni-Agustus 2018. Selama berkuliah, penulis juga menjadi administrator di Laboratorium Komputasi Berbasis Jaringan. Penulis dapat dihubungi melalui nomor *handphone* 082139497057 atau di email if.hania@outlook.com.

***(Halaman ini sengaja dikosongkan)***